POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Aerospaziale

Tesi di Laurea

Deep Learning for prediction of Aerodynamic Simulations

Politecnico di Torino

Relatore
Prof.ssa Sandra Pieraccini

Nicolò Crichigno

Luglio 2021

Abstract

The motivation of this work is to conjugate the potential of Artificial Neural Networks and the Aerodynamic field.
Artificial Neural Networks (ANNs) are a constantly growing field of study, everyday new applications in different fields such as Engineering, Medical, Data Science, or even 'Everyday Life' are found.
On the other hand, Computational Fluid Dynamic (CFD) is the 'state of the art' of the Aerodynamic analysis field, nowadays there's plenty of CFD software, both commercial and open-source, capable of simulating the most complex fluid model and geometry.
The goal is to develop a Neural Network (in this specific case a Multi-Layer Perceptron) that can predict the results of CFD simulations of a 2D airfoil giving as input three parameters: the Angle of Attack of the airfoil with respect to the free-stream velocity, the Mach number and the Reynolds number.
Two Neural Networks will be implemented: first, a Multi-Layer Perceptron will be trained to predict the 2 aerodynamic coefficients CL (Lift coefficient) and CD (Drag coefficient) of the airfoil. A second Multi-Layer Perceptron will be trained to predict the velocity field of the fluid around the airfoil.
The thesis is organized as follows. In **Chapter 1** there will be a brief introduction to the state of the art of the Computational Fluid Dynamic and Artificial Neural Networks and a discussion of the focus of this project and its possible applications. The **Chapter 2** consists of a theoretical analysis of the major aspects of a CFD analysis (set of differential equations, turbulence models and discretization models) and in **Chapter 3** the theoretical analysis of ANN's, their features and how the training process works will be treated.
After this introductory chapters, in **Chapter 4** the configuration of the CFD simulations used to produce the database and the features and architecture of the ANN's that was trained to predict the results of the simulations are presented and discussed. Finally, in **Chapter 5**, the results of the CFD simulations and the results obtained with the Neural Networks will be reported and discussed.

Acknowledgements

Vorrei ringraziare innanzitutto la mia relatrice, la professoressa Sandra Pieraccini, per avermi permesso di intraprendere questo percorso di tesi su un argomento tanto attuale quanto profondamente stimolante ed interessante, la ringrazio inoltre per la pazienza e l'infinita disponibilità nell'aiutarmi nei momenti di difficoltà.
Ringrazio inoltre il professor Domenic d'Ambrosio per essersi reso disponibile ad un utile e proficuo confronto nell'ambito della Fluidodinamica Computazionale.

Un ringraziamento va alla mia famiglia, a mia mamma Donatella, a mio papà Luciano, a mio fratello Jacopo e alla nonna Lina che in questi anni mi hanno supportato in ogni modo possibile, con sacrifici ma anche con grande fiducia nelle mie capacità.
Senza di loro questo traguardo sarebbe rimasto solamente un sogno lontano, queste quattro righe non sono che una briciola della gratitudine che dovrei dimostrar loro.

Ovviamente tra i 'Grazie' non possono mancare i miei amici, tutti coloro con cui ho legato in questi anni e con cui ho trascorso momenti belli e momenti difficili ma che nell'insieme hanno contribuito a rendermi la persona che sono oggi.
Non posso quindi che ringraziare innanzitutto Andrea (Postino) con cui ho convissuto per 5 anni e che considero un fratello più che un amico, Dascia con cui ho condiviso alcuni dei momenti più belli della mia vita finora e che con cui ho sviluppato un rapporto più unico che raro, Luca che con molta pazienza, disponibilità e gentilezza ha messo a disposizione la sua competenza e il suo tempo per aiutarmi a districarmi tra le insidie delle reti neurali, Stefano che in molte occasioni è stato un mentore oltre che un amico e con loro Pacio, Claudia, Fabio, Pronky, Momo, Roccino, Irene, Jessica, Simon, Scapa, Leonardo, Stefanino e tutti coloro che hanno lasciato una parte di loro nei miei ricordi e nella mia persona.

Approfitto infine di questo spazio per ringraziare HPC@POLITO per aver fornito le risorse di calcolo necessarie alla realizzazione di questo progetto di tesi.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal for this introductory chapter is to give a preliminary description of the two major subjects of this thesis project: the Computational Fluid Dynamic and the Artificial Neural Networks.

1.1 Introduction to Computational Fluid Dynamic

Computational Fluid Dynamic (CFD) is a branch of fluid mechanics that uses numerical analysis to numerically approximate problems involving a fluid flow [68].
CFD is a widely interdisciplinary subject that involves Mathematics, Fluid dynamics, Computer science and Data handling, it's used to obtain accurate results of physical phenomena more cost-efficiently with respect to experimental testing. Moreover, it enables to simulate flow around large-scale models that would be impossible to analyze through the experimental approach.
CFD presents several advantages:

- The cost of analysis is low compared to experiments.
- CFD analysis requires less time to get results compared to experimental tests.
- CFD allows to get detailed information of the flow at every point of the domain, in an experimental test there's no possibility to place pressure ports and probes at all locations.

In Figure 1.1 we present a comparison between the cost of the CFD simulations (old techniques and modern techniques) and the experimental approach (wind tunnel experiments). Improvements in meshing strategy translated down the older CFD curve leading to an overall diminishing of basic cost per simulation, meanwhile new algorithms and better computational hardware led to a decrease of the slope of the curve, resulting in an improvement of cost-efficiency [46].

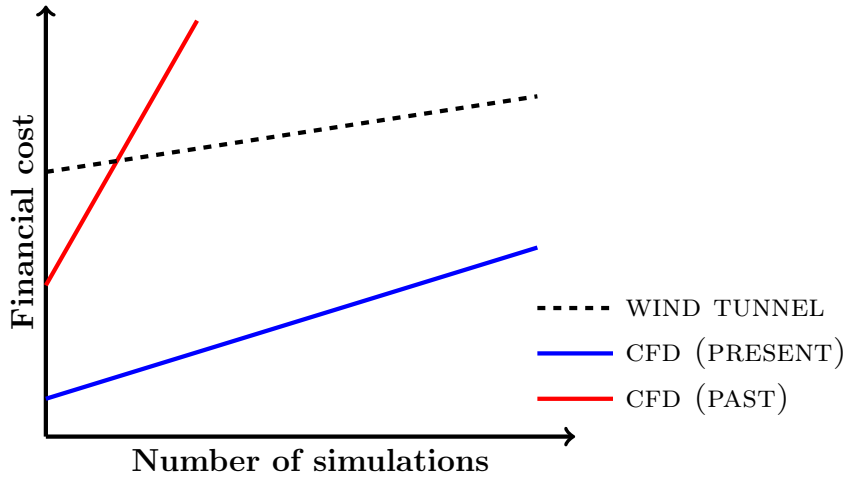

Figure 1.1. Cost-efficiency comparison between CFD (past and present) and experimental approach

Components of a CFD Simulation

Given a fluid dynamic problem, the process behind the definition of a CFD analysis can be divided into three major 'steps'.

Pre-processing

Pre-processing is the first step in CFD simulation, which is necessary to correctly define the parameters of the simulation, it can be subdivided into different sub-steps:

- **Problem Analysis**: Problem analysis is the cornerstone of the simulation; it's necessary to understand the problem that has to be solved in order to properly define the objectives and parameters of the CFD simulation.

- **Geometry**: Once the physics of the problem is defined the next step is to define a geometry (which can be 2D or 3D, depending on the problem setting) that correctly represents the problem.

- **Mesh**: The computational grid, or 'mesh', is a key factor of a CFD simulation, it affects every other aspect of it and heavily contributes to the final quality of the results. The governing equations will be solved, and the fluid properties evaluated, for every cell of the mesh, giving the representation of the flow in the physical domain. A finer mesh, with several small cells, leads to more accurate results but with an increase in the computational cost of the simulation.

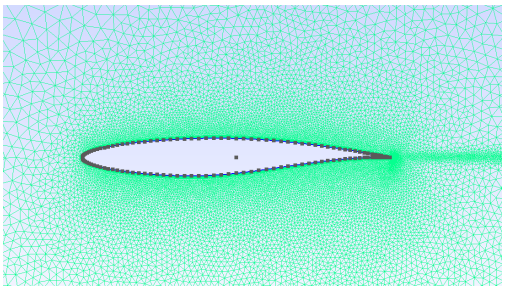

Figure 1.2. Mesh around an airfoil (software: Gmsh)

- **Setup of the solver**: This stage consists in the definition of the flow conditions of the problem into the software: boundary conditions, flow type and flow properties, mathematical models for the equations, turbulence models, discretization schemes, etc.

Processing

Once all the pre-processing steps have been undertaken, it's possible to run the simulations on a CFD software. At this stage, the software will iteratively solve the system of equations in order to obtain the lowest residual (which can be considered the 'error' of the CFD simulation) which is computed based on the solver setup. When the 'Convergence criteria' is satisfied the processing ends and the results are obtained.

Post-processing

When the results are obtained the final step is to analyze this results to produce images, vector-plots, streamlines and data plots, the focus is to correctly represent every aspect of the solution.

The computational cost

Setting up the discrete system and solving it involves a very large number of repetitive calculations, a task that humans palm over to the digital computer [50]. The computational effort of a CFD simulation grows rapidly with it's complexity: high resolution models, unsteadiness in the flow field, multi-phase flows, etc., all these different aspects contribute to increase the computational cost of a simulation. This problem leads to different possible solutions.

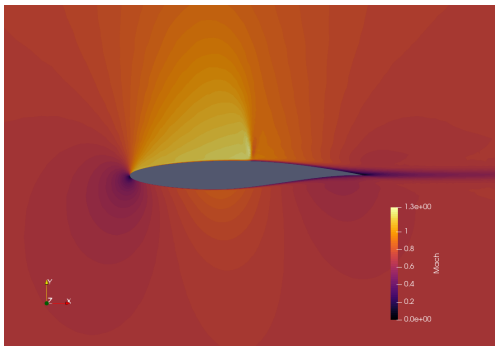

Figure 1.3. Mach field visualization around an airfoil (software: Paraview v5.9.0)

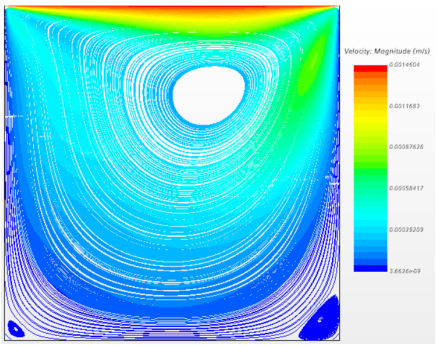

Figure 1.4. Streamlines and velocity field, lid driven cavity test-case (software: Siemens Simcenter STAR-CCM+ v.2019.3)

Increase the computational power available

High Performance Computing (HPC) is a well consolidated technology based on the concept of parallel computing, which consists in the partition of a high computational cost demanding task on many different CPUs.
There are different possible architectures for a HPC system, the most common is the 'cluster' version where all computing units, or 'nodes', work simultaneously on the same task, to do this all nodes are connected to each other using fast local area networks (LAN), for example the 'InfiniBand' communication standard which is widely used for node-to-node, node-to-storage or intra-node communication in many different supercomputers or cluster systems. To measure the performance of a HPC system the FLOPS (FLoating-point Operations Per Second) is used, the magnitude of the computational power of modern super computers is measured in 'peta-FLOPS' (or PFLOS, which is 10^{15} FLOPS).
Some examples of modern supercomputer systems (according to the 'TOP500' November 2020 list [2]) are:

- Supercomputer Fugaku (RIKEN Center for Computational Science, Japan) with a max computational speed of 442.01 PFLOPS
- IBM Power System AC922 'Summit' (DOE/SC/Oak Ridge National Laboratory, United States) with a max computational speed of 148.6 PFLOPS
- IBM Power System AC922 'Sierra'(DOE/NNSA/LLNL, United States) with a max computational speed of 96.64 PFLOPS

Simplification and optimization of the overall problem

Simulating a full scale turbulent flow in the time domain using the Direct Numerical Simulation (DNS) method, which means solving the full set of Navier-Stokes

equations for every element of the grid in every single time-step, would give as result an almost-exact depiction of the analyzed phenomena but, on the other hand, brings with it an overwhelming computational cost. The FLOPS needed to perform a turbulent flow DNS grows rapidly with the size and complexity of the computational grid and with the number of time-steps but it also grows with the cube of the Reynolds number of the phenomena, this could quickly overcome the maximum computational power of any supercomputer existing at this time.
For this reason several simplified and optimized models exist, instead of solving the exact phenomena different approximations are used in order to give an overall less accurate result but with a significant decrease in terms of computational cost. For example, for a turbulent problem like the aforementioned one, different approximate approaches exist:

- Raynolds Averaged Navier-Stokes (RANS), proposed by Osborne Reynolds, is a well consolidated model for solution of turbulent phenomena, it relies on the concept of 'Reynolds decomposition' which states that an instantaneous time-dependent quantity can be separated in its time-averaged component and fluctuating component. This set of approximate Navier-Stokes equations brings with it a drawback: after the decomposition and time-averaging of the original equations a non-linear term remains (the so called Reynolds stress tensor) that requires additional modelling equations for closing the RANS problem.
  In Chapter 2 we will dive much deeper into the RANS method, as it's the one used for the CFD simulations that will produce the dataset for the ANN, subject of this thesis.

- Large Eddies Simulation (LES), firstly proposed by Joseph Smagorinsky (1963) [59], is another approximate model for turbulent flow modeling. Since turbulence is a multi-scale phenomena, the basic concept of LES is to apply a spatial low-pass filter that ignores the smaller length-scales of the flow (the so-called 'sub-grid components', which are the most expensive to simulate in terms of computational cost). If the problem analyzed has a large characteristic dimension the smaller scales contribute to the overall flow can be treated statistically as a time and spatial average as it's quite irrelevant for the 'global' scale of the phenomena. On the other hand, if the smaller length-scales gives a relevant contribution to the overall phenomena (for example in near-wall flows or multi-phase flows) they need to be modeled (some sub-grid components models are Smagorinsky's model, Dynamic Smagorinsky's model or the sub-grid turbulent kinetic energy model).

Another recently developed simulation technique for turbulent flows is the Hybrid RANS/LES model which uses the RANS method to solve the smaller scale near-wall flow and the outer, larger scale, layer of the flow is modelled with the LES approach.

Pre-simulation on a coarser grid model

If the target of the simulation is a flow with particular relevant behaviours or interactions that involve high gradients, fast change in flow quantities etc., these behaviours need particular attention and a higher resolution of the grid to be correctly represented. The problem rises when it's not known 'a-priori' where (or when, if the phenomena is time dependant) these behaviours could occur in the simulation. For example, if the CFD simulation involves a transonic flow around an airfoil, the presence of a shock-wave on the upper-side of the airfoil is expected, but the exact location is not known in advance, because it depends on many different factors, and refining entirely the mesh to be sure that the shock will be accurately depicted would lead to an unfeasible increase in the computational cost.
A possible solution to this problem is to run a preliminary simulation of the flow on a quite coarse grid, this gives a good 'picture' of the overall flow. It's then possible to use the results of this coarser solution to compute the gradients or the second-order space derivative (as shown in fig. 1.5 and 1.6) of a certain quantity that is representative of the flow (for example, in an airfoil simulation, the local flow velocity or the static pressure).

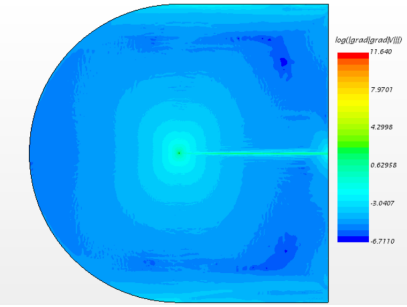

Figure 1.5. Second-order space derivative of the local flow velocity (Software: Siemens Simcenter STAR-CCM+ v.2019.3 [4])

Figure 1.6. Second-order space derivative of the local flow velocity, detail of the flow around the airfoil

It's now possible to define a 'mesh refining function' based on these gradients/second derivatives of the coarse solution that refines the mesh locally only in the areas that present higher gradients in order to increase the accuracy of the simulation only where it's needed, this leads to an extremely accurate depiction of the flow with a limited increase in terms of computational effort. Some software, like Siemens Simcenter STAR-CCM+ [4], gives also the possibility to define the solution on the refined grid extrapolating it from the coarse one through an interpolation process, with a further save of computational resources.
Another possible way to obtain this preliminary simulation is through Artificial Intelligence. If we train an Artificial Neural Network to learn, for example, the velocity field around an airfoil using many different CFD simulations results as input database, it's then possible to use the predictions of the velocity field in different free-stream conditions as a preliminary result to compute the gradients of the velocity fields and finally obtain the optimally refined mesh, saving a lot of time and computational cost but still obtaining the desired accuracy for the CFD simulation.

1.2 Introduction to Artificial Neural Networks

The concept of Machine Learning (ML) firstly appeared in the early of the XX century, it's based on the concept of using statistical methods to improve the performance of pattern recognition and prediction [10].
Machine Learning methods can be split in 3 main categories, also known as 'paradigms':

- **Unsupervised Learning**: The neural network need to learn a pattern from the inputs without having any output as reference target.

- **Supervised Learning**: The neural network receives both inputs and desired target outputs and needs to define a pattern that successfully associates the inputs and the outputs.

- **Reinforced Learning**: In this case, the model interacts with a dynamic environment in which it must achieve a certain goal (such as driving a vehicle or playing a game against an opponent).
  As it navigates its problem space, the program is provided feedback that's analogous to a reward, which it tries to maximize [10].

Artificial Neural Networks (ANNs) are a subset of the Supervised Learning methods. The basic principles behind the supervised training process of an ANN are the following: the network first creates a mathematical model based on the inputs it receives (this is called 'forward' process or 'inference'), then it computes the classification error (or 'loss') between the desired and the predicted outputs. This error is then used in the so-called 'backward' process to update and adjust the previously made model by modifying its internal parameters.

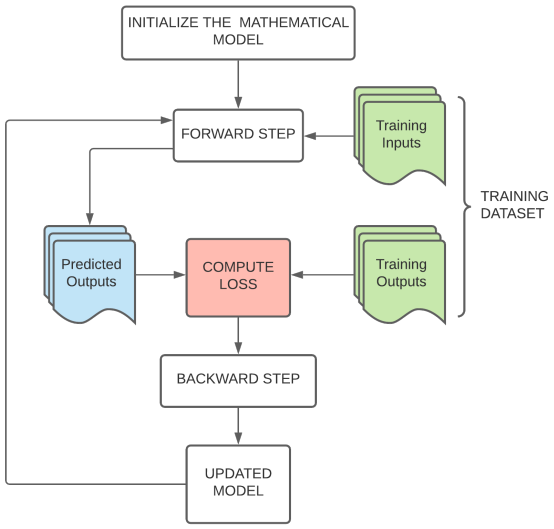

Figure 1.7. Flow-chart of a basic training process of an ANN

The architecture of an ANN loosely reminds the structure of the animal brain: the basic unit of a neural network is the 'Neuron' which is closely linked to the others units of the network in different and complex ways based on the specific goal it has to achieve.
As mentioned, the neuron is the fundamental element of an ANN and it's responsible for all the basic operations, they behave in different ways, depending on the model of the neuron. Each neuron model is characterized by an activation function that decides if the neuron is activated or inactivated based on the input it receives. ANNs consist of various groups of neurons organized in different ways, the basic structure of organized neurons is called 'Layer', many different layers can be present in a neural network. The layers of a neural network can be generically divided into 3 main categories based on the specific elaboration step of the layer (fig. 1.8):

- **Input layer**: It receives the input from the training dataset

- **Hidden layer**: It elaborates the data received from the Input layer

- **Output layer**: It gives the results of the computation

The architecture depicted above is just the basic structure of a neural network, it can be expanded by adding more hidden layers and increase the complexity of the layers themselves.
Based on the complexity of the mathematical model that defines the Neuron behaviour, we can classify neural networks into three generations [43].

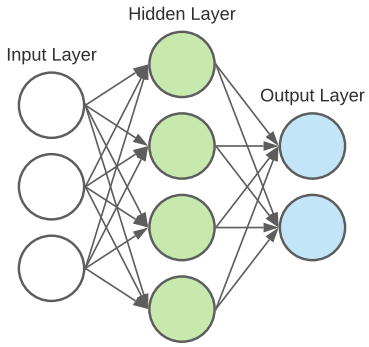

Figure 1.8. Basic structure of the layers in a neural network

First generation Artificial Neural Networks

The first generation of ANNs, which are also called 'Perceptrons' or 'Thresholdgates') is based on the concept of the neuron of McCulloch-Pitts:

$$y = \begin{cases} 1, & \text{if } h = \sum_i x_i w_i \geq u \\ 0, & \text{otherwise} \end{cases} \tag{1.1}$$

where y is the output of the neuron, that can be 0 (stands for not activate) and 1 (stands for activate), h is the state of the neuron, w_i is a synaptic weight, x_i is an output of the previous neurons and u is the threshold, that at the beginning was set to 0, and then can be set by the user (introduction of a bias), giving more flexibility to the network [69].
A Perceptron doesn't work well with continuous data, in fact is an algorithm used for supervised learning of binary classifiers which are a type of networks that decide whether an input, usually represented by a series of vectors, belongs to a specific class.
In short, a perceptron is a single-layer neural network that consists of four main parts (fig. 1.9): input values, weights and bias, net sum, and an activation function (which is a step function) [3].
Thanks to the binary nature of this ANN, the theory of Hebb[1] is used for training this type of NN: it claims that if a pre-synaptic neuron (that acts as input) causes

[1]Donald Olding Hebb (July 22, 1904 – August 20, 1985) was a Canadian psychologist that worked in the neurophysiology field.
He is popular for the Hebbian rule, which asserts that if a pre-synaptic cell helps to fire a post-synaptic cell, the synapses is strengthened [32].

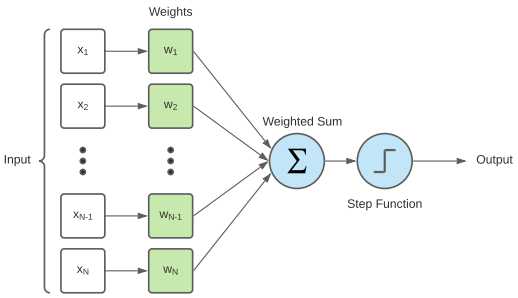

Figure 1.9. Perceptron

the activation of the post-synaptic neuron (it receives the input stimulus), the synaptic weight is enhanced [69].

Second generation of Artificial Neural Networks

The second generation of ANNs is an evolution of the previous one, introducing a continuous activation function.
The neuron can be represented as:

$$y = g(\sum_i x_i w_i - b) \tag{1.2}$$

where g is the activation function (different functions are possible, Fig. 1.10: Sigmoid, Hyperbolic tangent, ReLU, etc.) and b is a translation term called 'Bias'.

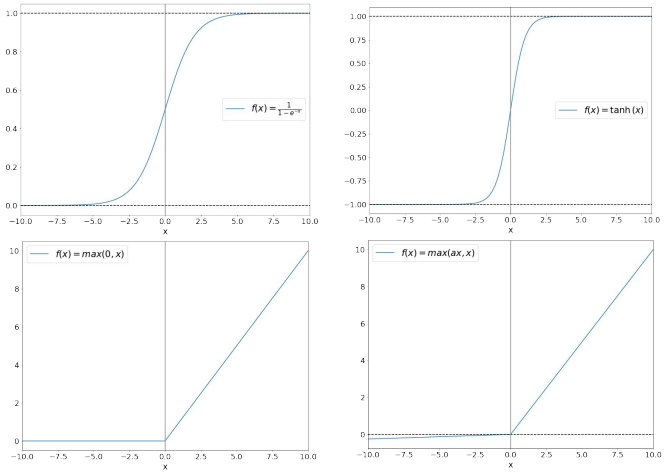

Figure 1.10. Different types of activation function g

With this type of ANNs the training process works differently, the rule of Hebb becomes outdated and new types of algorithms are employed, such as the family of algorithms based on the concept of gradient descendent, which allows to update the internal parameters of the neurons in the layers using the error between the predicted output and the expected one. These methods allows the training of ANNs containing many hidden layers (Deep Learning) [69].
The second generation of ANNs is computationally more powerful: it has been proven that an ANN of second generation can approximate any continuous function with just one hidden layer [69].

Third generation of Artificial Neural Networks

The third generation neural networks introduce a new type of neural model called Spiking Neuron. The model behind this new type of neurons is based on a system of Ordinary Differential Equation (ODE) and simulate the spike dynamics of the biological neuron.

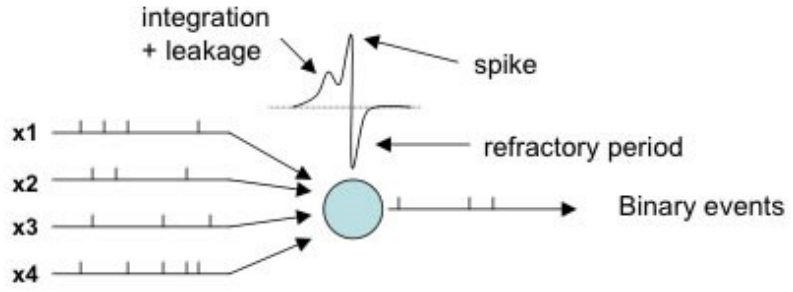

Figure 1.11. Spiking Neuron model [13]

The presence of the spikes causes the information to be encoded over time: the codification is determined by the firing rate or by the period of time between two consequent spikes [41].
This type of ANN can learn any function can be learned by the first two generations of ANN using a single hidden layer and, in some cases, with fewer neurons [69].

Chapter 2

Theoretical analysis: Computational Fluid Dynamics

In this chapter the focus will be the theory behind the main aspects and features of a CFD simulation, with particular focus on the methods, models and features chosen for the definition of the RAE 2822 Transonic Airfoil Simulation (which will be discussed later in this document).
As anticipated in section 1.1, Computational Fluid Dynamics is a widely interdisciplinary subject, in the following sections the main elements that compose a CFD simulation will be described from a theoretical point of view, namely:

- System of Governing Equations: Derivation of the Reynolds Averaged Navier-Stokes (RANS) system of Partial Differential Equations (PDE) starting from a discussion on the set of Navier-Stokes equations.
- Turbulence Model: Analysis of different Turbulence models used in modern CFD, in particular the Spalart-Allmaras model.
- Mesh: Description of the different features of a computational grid.
- Discretization Schemes: Introduction on the possible discretization methods for the system of PDE, in particular the Jameson-Schmidt-Turkel (JST) Scheme.

2.1 Governing Equations

Navier-Stokes Equations

The Navier-Stokes Equations describe how the velocity, pressure, temperature, and density of a moving fluid are related. The equations were derived independently by George Gabriel Stokes, in England, and Claude-Luis Navier, in France, in the early 1800's. They are an extensions of the Euler Equations and include the effects of viscosity on the flow [15].
The Navier-Stokes Equations consist of a time-dependent continuity equation for conservation of mass, three time-dependent conservation of momentum equations and a time-dependent conservation of energy equation.
There are four independent variables in the problem: the x, y, and z spatial coordinates of the computational domain, and the time t.
There are six dependent variables: the pressure p, density ρ, and temperature T (which is contained in the energy equation through the total energy E) and three components of the velocity vector (the u component is in the x direction, the v component is in the y direction, and the w component is in the z direction). All the dependent variables are functions of all four independent variables, therefore the differential equations are Partial Differential Equations (PDE) [15].
In the following the Navier-Stokes Equations for a compressible, 3 dimensional and unsteady fluid will be presented in their differential and conservative formulation, which means that they are written for an infinitesimal control volume fixed in space.

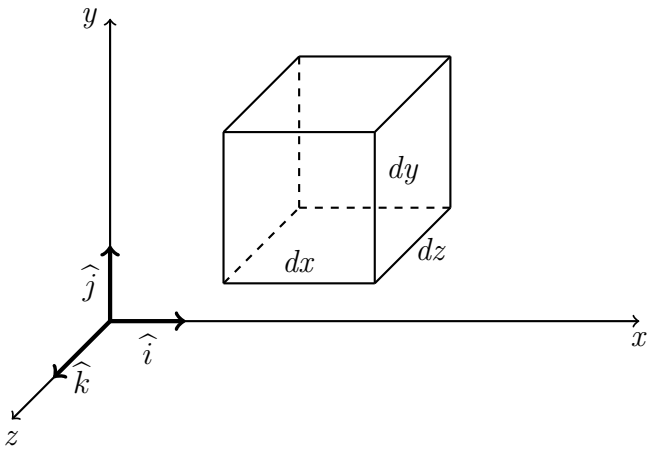

Figure 2.1. Infinitesimal control volume, dimensions: dx, dy, dz

Continuity Equation

The continuity equation (or mass balance equation) states that the rate of change of the mass inside the control volume (fig. 2.1) is equal to the net flux of the mass coming out of the volume through its surfaces.

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = 0 \tag{2.1}$$

The equation consists in the time variation of the density inside the control volume and the convective transport of mass through the volume surfaces.

Momentum Balance Equations

The momentum balance equations form a system of three equations related to the x, y and z directions. It's the result of the application of the Second Newton's Law in the 3 directions to the control volume, upon which pressure and friction forces act (only the surface forces acting on the control volume are considered, the volume forces in this formulation are neglected).

x-direction

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} + \frac{\partial(\rho uw)}{\partial z} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left[\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right] \tag{2.2}$$

y-direction

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho vu)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} + \frac{\partial(\rho vw)}{\partial z} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left[\frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right] \tag{2.3}$$

z-direction

$$\frac{\partial(\rho w)}{\partial t} + \frac{\partial(\rho wu)}{\partial x} + \frac{\partial(\rho wv)}{\partial y} + \frac{\partial(\rho w^2)}{\partial z} = -\frac{\partial p}{\partial z} + \frac{1}{Re}\left[\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right] \tag{2.4}$$

The LHS accounts for the time variation of momentum and the convective transport of momentum through the volume surfaces; The RHS collects pressure forces and friction forces. The latter term is a contribute related to the viscosity of the fluid and the presence of velocity gradients; in fact, for a Newtonian fluid, the shear stresses can be expressed as:

$$\tau_{ij} = \delta_{ij}\lambda \nabla \cdot \vec{V} + \mu\left(\frac{\partial u_i}{x_j} + \frac{\partial u_j}{x_i}\right) \tag{2.5}$$

where τ_{ij} represents the stress in the j-direction that acts on a surface whose normal unit vector points to the i-direction and δ_{ij} is the so-called 'Kronecker's delta' which is 1 if $i = j$ (normal shear stress) and 0 if $i \neq j$ (tangential stress), μ is the dynamic

viscosity of the fluid and λ is the so-called 'bulk viscosity' which can be expressed, in aerodynamic applications, as $\lambda = -\frac{2}{3}\mu$.
The importance of the viscous term in (2.2), (2.3) and (2.4) is dictated by the inverse of the **Reynolds number**:

$$Re = \frac{\rho V L}{\mu} \tag{2.6}$$

This dimensionless term represents the ratio between the contributes of the convection and diffusion, a high Reynolds number means that convection is predominant in respect to diffusion as transport phenomena.

Energy Balance Equation

The energy balance equations is obtained by applying the First Principle of Thermodynamics on the control volume. According to such principle, the rate of change of total energy inside the control volume is equal to the sum of the net heat flux through the volume and the work done per unit time by the surface (and volume, if present) forces on the control volume.

$$\begin{aligned}\frac{\partial(\rho E)}{\partial t} + \frac{\partial(\rho u E)}{\partial x} + \frac{\partial(\rho v E)}{\partial y} + \frac{\partial(\rho w E)}{\partial z} = &-\left[\frac{\partial(up)}{\partial x} + \frac{\partial(vp)}{\partial y} + \frac{\partial(wp)}{\partial z}\right] \\ &-\frac{1}{Re\ Pr}\left[\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z}\right] \\ +\frac{1}{Re}\left[\frac{\partial}{\partial x}(u\tau_{xx} + v\tau_{xy} + w\tau_{xz}) + \frac{\partial}{\partial y}(u\tau_{yx} + v\tau_{yy} + w\tau_{yz}) + \frac{\partial}{\partial z}(u\tau_{zx} + v\tau_{zy} + w\tau_{zz})\right]\end{aligned} \tag{2.7}$$

The LHS of the equations presents the time variation of the total energy in the control volume and the convective transport of total energy; the RHS contains, left to right: the work done per unit time by the pressure forces, the net heat flux and the work done by the shear stresses.
The RHS of (2.7) contains another dimensionless term, the **Prandtl number**:

$$Pr = \frac{\mu c_p}{k} = \frac{\nu}{\alpha} \tag{2.8}$$

This term represents the ratio between cinematic diffusion (ν) and thermal diffusion (α).
The total energy per unit mass, E, can be expressed as the sum of the internal energy per unit mass of the fluid and its kinetic energy per unit mass:

$$E = e + \frac{1}{2}|\vec{V}|^2 \tag{2.9}$$

For a perfect gas, with specific heats constant c_v and c_p, the internal energy can be written as $e = c_v T$, in this way the total energy of the fluid in the control volume is only function of the temperature and fluid velocity, which are both unknown dependant variables of the governing equations.
The second term on the RHS contains the heat fluxes due to heat conduction and they depend on the temperature gradients through the Fourier's law:

$$\dot{q}_i = -k\frac{\partial T}{\partial x_i} \tag{2.10}$$

with k thermal conductivity of the fluid.

Since the CFD simulation that is the object of this thesis project involves a transitional/fully turbulent flow, it's necessary to introduce a model of governing equations suitable for the solution of turbulent flows and which doesn't involve an unbearable computational cost. This set is the RANS (Reynolds-Averaged Navier-Stokes) system of equations.

Reynolds-Averaged Navier Stokes Equations

The Reynolds-averaged Navier–Stokes equations are time-averaged equations of motion for fluid flow, the basic concept behind the equations is the 'Reynolds decomposition': any instantaneous quantity can be decomposed into its time-averaged and fluctuating quantities, this concept was firstly proposed by Osborne Reynolds. The RANS equations are primarily used to describe turbulent flows, they can be used with approximations based on knowledge of the properties of flow turbulence to give approximate time-averaged solutions to the Navier–Stokes equations [51].

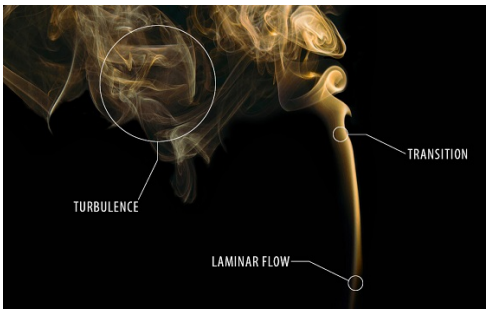

Figure 2.2. Transition from laminar to turbulent flow [45]

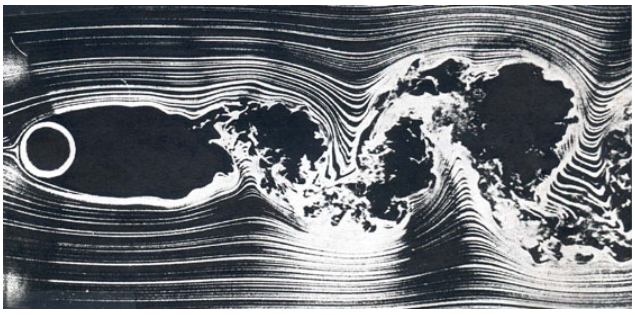

Figure 2.3. Turbulent wake behind a cylinder [26]

Turbulent flows are characterized by self sustained velocity, pressure and density fluctuations superimposed to the main flow. The intensity of fluctuations is variable, but it usually doesn't exceed a value between 10% to 30% of the mean value of the considered quantity. Looking at the time history of a velocity component

at a given point, for example, it looks like a random signal, but it is not really random, because there is a recognizable structure in turbulence.
The irregularities in the velocity field can look like a vortex, or a jet, or any other recognizable form, these irregularities are called 'eddies'. Eddies are embedded one in each other, small eddies can exist inside larger eddies and even smaller eddies can be found inside the former, this is related to the multi-scale nature of the turbulence. This continuous distribution of the eddies size characterize turbulence with respect, for example, to a separated laminar flow where large scale vortices detach from a body, in this case the irregularity in the flow would be limited to just a few frequencies.

Reynolds Decomposition

As previously mentioned, the basic idea behind the derivation of RANS model is the Reynolds decomposition.

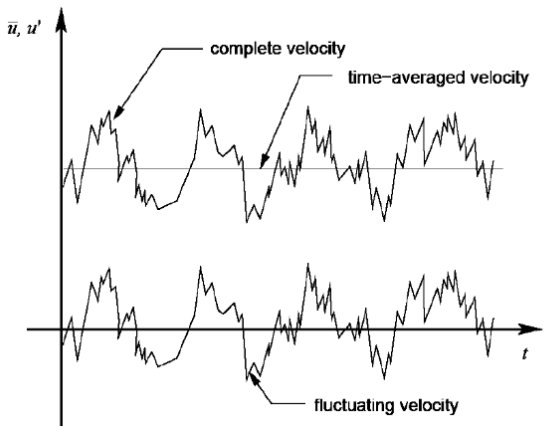

Figure 2.4. Reynolds decomposition [35]

Given a physical quantity $u(x, y, z, t)$, the Reynolds decomposition states that the instantaneous value of this quantity can be expressed as the sum of its averaged value (space, time or ensemble average) and the deviation from it (the so-called 'fluctuation').

$$u(x, y, z, t) = \overline{u}(x, y, z) + u'(x, y, z, t) \tag{2.11}$$

where $\overline{u}(x, y, z)$ is the averaged value (time average, in this case) and $u'(x, y, z, t)$ the fluctuation.
Depending on the type of turbulence that needs to be modeled there are three possible ways to obtain the averaged value [49].

- **Time Average**: Suitable for turbulence phenomena where the mean value doesn't change in time (Stationary turbulence).

$$\overline{u}(x,y,z) = \lim_{T\to\infty} \frac{1}{T} \int_t^{t+T} u(x,y,z,t)\, dt \tag{2.12}$$

  where T is the averaging time period.

- **Space Average**: This averaging method is used for turbulence flow that are uniform in all directions (Homogeneous turbulence).

$$\overline{u}(t) = \lim_{V\to\infty} \frac{1}{V} \iiint_V u(x,y,z,t)\, dV \tag{2.13}$$

  where V is the averaging volume.

- **Ensemble Average**: Used for flows that decay in time, ensemble average is made upon N identical experiments on the same flow and conditions.

$$\overline{u}(x,y,z,t) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} u_n(x,y,z,t) \tag{2.14}$$

It is clear from the definition of the averaging operation that the Reynolds decomposition possesses the following two properties [40]:

$$\overline{\overline{u}} = \overline{u} \qquad\qquad \overline{u'} = 0 \tag{2.15}$$

These two properties are fundamental for the derivation of the RANS equations.

Derivation of RANS Equations

Let's consider a 3 dimensional turbulent flow, for the sake of simplicity is considered a flow where density and temperature don't change in time.
In this case the instantaneous flow variables can be expressed, using the Reynolds decomposition, as the sum of their average value and fluctuation:

$$\begin{aligned} u &= \overline{u} + u' \\ v &= \overline{v} + v' \\ w &= \overline{w} + w' \\ p &= \overline{p} + p' \end{aligned} \tag{2.16}$$

Because changes in density are considered negligible (which is an approximation valid only for a flow with low Mach number, specifically $M < 0.3$), the fluid evolution can be described using the incompressible formulation of the Navier-Stokes

equations.

Continuity Equation

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{2.17}$$

Momentum Balance Equation (x-direction)

$$\rho \left[\frac{\partial u}{\partial t} + \frac{\partial (u^2)}{\partial x} + \frac{\partial (uv)}{\partial y} + \frac{\partial (uw)}{\partial z} \right] = -\frac{\partial p}{\partial x} + \mu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right] \tag{2.18}$$

The others direction can be written in the same way.
Starting from the **Continuity Equation** and applying the Reynolds decomposition:

$$\frac{\partial (\overline{u} + u')}{\partial x} + \frac{\partial (\overline{v} + v')}{\partial y} + \frac{\partial (\overline{w} + w')}{\partial z} = 0$$

$$\frac{\partial \overline{u}}{\partial x} + \frac{\partial u'}{\partial x} + \frac{\partial \overline{v}}{\partial y} + \frac{\partial v'}{\partial y} + \frac{\partial \overline{w}}{\partial z} + \frac{\partial w'}{\partial z} = 0$$

Assuming a stationary turbulence, it's possible to time-average the decomposed equation:

$$\overline{\frac{\partial \overline{u}}{\partial x} + \frac{\partial u'}{\partial x} + \frac{\partial \overline{v}}{\partial y} + \frac{\partial v'}{\partial y} + \frac{\partial \overline{w}}{\partial z} + \frac{\partial w'}{\partial z}} = 0$$

$$\overline{\frac{\partial \overline{u}}{\partial x}} + \overline{\frac{\partial u'}{\partial x}} + \overline{\frac{\partial \overline{v}}{\partial y}} + \overline{\frac{\partial v'}{\partial y}} + \overline{\frac{\partial \overline{w}}{\partial z}} + \overline{\frac{\partial w'}{\partial z}} = 0$$

Now, using the average operation properties in (2.15), it's possible to set to 0 the mean values of the fluctuations, obtaining the **Time-averaged Continuity Equation**:

$$\frac{\partial \overline{u}}{\partial x} + \frac{\partial \overline{v}}{\partial y} + \frac{\partial \overline{w}}{\partial z} = 0 \tag{2.19}$$

It's easy to notice that the time-averaged continuity equation is identical in its formulation to the normal continuity equation with the instantaneous values replaced by the time-averaged ones.
The next step is to derive the time-averaged momentum balance equation, the process is the same used previously. Since the equation is longer and more complex the LHS and the RHS will be treated separately.

LHS: Applying the Reynolds decomposition an then time-averaging the equation obtained:

$$LHS = \rho \left[\underbrace{\frac{\partial u}{\partial t}}_{\text{1st term}} + \underbrace{\frac{\partial (u^2)}{\partial x}}_{\text{2nd term}} + \underbrace{\frac{\partial (uv)}{\partial y}}_{\text{3rd term}} + \underbrace{\frac{\partial (uw)}{\partial z}}_{\text{4th term}} \right] \tag{2.20}$$

1st term:

$$\frac{\partial(\overline{u}+u')}{\partial t}=\frac{\partial\overline{u}}{\partial t}+\frac{\partial u'}{\partial t} \tag{2.21}$$

Note that, even since the mean velocity $\overline{u}$ is independent of the time-averaging period, it's not possible to simply set equal to 0 its time derivative because the global flow is intrinsically unsteady and its properties can still change in time. Therefore two different timescales have to be distinguished, one for the turbulence (T_1) and one for the global flow (T_2), and choose the averaging time such that $T_1 < T < T_2$ so that only the turbulent fluctuations are time-averaged. Hence, the time derivative term for the mean velocity is not eliminated [49].
Now time-averaging the (2.21):

$$\overline{\frac{\partial\overline{u}}{\partial t}+\frac{\partial u'}{\partial t}}=\overline{\frac{\partial\overline{u}}{\partial t}}+\overline{\frac{\partial u'}{\partial t}}$$

and applying the average operation properties (2.15), the 1st term of the LHS becomes:

$$\overline{\frac{\partial\overline{u}}{\partial t}+\frac{\partial u'}{\partial t}}=\overline{\frac{\partial\overline{u}}{\partial t}}=\frac{\partial\overline{u}}{\partial t} \tag{2.22}$$

The **2nd**, **3rd** and **4th terms** of the LHS are the convective terms of the equations, the steps are the same:

$$\frac{\partial[(\overline{u}+u')^2]}{\partial x}=\frac{\partial}{\partial x}(\overline{u}^2+2\overline{u}u'+u'^{\,2}) \tag{2.23}$$

$$\overline{\frac{\partial}{\partial x}(\overline{u}^2+2\overline{u}u'+u'^{\,2})}=\overline{\frac{\partial}{\partial x}(\overline{u}^2+u'^{\,2})}+\overline{\frac{\partial}{\partial x}(2\overline{u}u')}$$

Since $\overline{u'}=0$, consequently also $\frac{\partial}{\partial x}(2\overline{u}u')=0$. Therefore, the time average of the second term is:

$$\overline{\frac{\partial[(\overline{u}+u')^2]}{\partial x}}=\frac{\partial}{\partial x}(\overline{\overline{u}^2+u'^{\,2}})=\frac{\partial}{\partial x}(\overline{u}^2+\overline{u'^{\,2}}) \tag{2.24}$$

The 3rd and 4th terms are obtained in the same way, so the results are:

$$\overline{\frac{\partial[(\overline{u}+u')(\overline{v}+v')]}{\partial y}}=\frac{\partial}{\partial y}(\overline{u}\overline{v}+\overline{u'v'}) \tag{2.25}$$

$$\overline{\frac{\partial[(\overline{u}+u')(\overline{w}+w')]}{\partial z}}=\frac{\partial}{\partial z}(\overline{u}\overline{w}+\overline{u'w'}) \tag{2.26}$$

In conclusion, the **LHS** of the time-averaged momentum balance equation is:

$$LHS=\rho\left[\frac{\partial\overline{u}}{\partial t}+\frac{\partial}{\partial x}(\overline{u}^2+\overline{u'^{\,2}})+\frac{\partial}{\partial y}(\overline{u}\overline{v}+\overline{u'v'})+\frac{\partial}{\partial z}(\overline{u}\overline{w}+\overline{u'w'})\right] \tag{2.27}$$

RHS : The right-hand side of the x-direction momentum balance equation is:

$$RHS = \underbrace{-\frac{\partial p}{\partial x}}_{\text{Pressure term}} + \underbrace{\mu\left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right]}_{\text{Shear-stress term}} \tag{2.28}$$

The **pressure gradient term** becomes:

$$-\overline{\frac{\partial(\overline{p}+p')}{\partial x}} = -\left(\overline{\frac{\partial \overline{p}}{\partial x} + \frac{\partial p'}{\partial x}}\right) = -\left(\overline{\frac{\partial \overline{p}}{\partial x}} + \overline{\frac{\partial p'}{\partial x}}\right) = -\frac{\partial \overline{p}}{\partial x} \tag{2.29}$$

The **Shear-stress term** can be decomposed as:

$$\mu\left[\frac{\partial^2}{\partial x^2}(\overline{u}+u') + \frac{\partial^2}{\partial y^2}(\overline{u}+u') + \frac{\partial^2}{\partial z^2}(\overline{u}+u')\right]$$

Time-averaging the decomposed term:

$$\mu\left[\overline{\frac{\partial^2}{\partial x^2}(\overline{u}+u') + \frac{\partial^2}{\partial y^2}(\overline{u}+u') + \frac{\partial^2}{\partial z^2}(\overline{u}+u')}\right]$$

$$\mu\left[\frac{\partial^2}{\partial x^2}(\overline{\overline{u}}+\overline{u'}) + \frac{\partial^2}{\partial y^2}(\overline{\overline{u}}+\overline{u'}) + \frac{\partial^2}{\partial z^2}(\overline{\overline{u}}+\overline{u'})\right]$$

Finally, applying again the properties in (2.15) the **Shear-stress term** becomes:

$$\mu\left[\frac{\partial^2 \overline{u}}{\partial x^2} + \frac{\partial^2 \overline{u}}{\partial y^2} + \frac{\partial^2 \overline{u}}{\partial z^2}\right] \tag{2.30}$$

Therefore, the **RHS** of the time-averaged momentum equation is:

$$RHS = -\frac{\partial \overline{p}}{\partial x} + \mu\left[\frac{\partial^2 \overline{u}}{\partial x^2} + \frac{\partial^2 \overline{u}}{\partial y^2} + \frac{\partial^2 \overline{u}}{\partial z^2}\right] \tag{2.31}$$

In conclusion, putting together the LHS (2.27) and the RHS (2.31), the **time-averaged momentum balance equation** in the x-direction is obtained:

$$\begin{aligned}&\rho\left[\frac{\partial \overline{u}}{\partial t} + \frac{\partial}{\partial x}(\overline{u}^2 + \overline{u'^{\,2}}) + \frac{\partial}{\partial y}(\overline{uv} + \overline{u'v'}) + \frac{\partial}{\partial z}(\overline{uw} + \overline{u'w'})\right] = \\ &-\frac{\partial \overline{p}}{\partial x} + \mu\left[\frac{\partial^2 \overline{u}}{\partial x^2} + \frac{\partial^2 \overline{u}}{\partial y^2} + \frac{\partial^2 \overline{u}}{\partial z^2}\right]\end{aligned} \tag{2.32}$$

It's very important to do some further rearrangements to the terms of the equation:

$$\begin{aligned}&\rho\left[\frac{\partial \overline{u}}{\partial t} + \frac{\partial}{\partial x}(\overline{u}^2) + \frac{\partial}{\partial y}(\overline{uv}) + \frac{\partial}{\partial z}(\overline{uw})\right] = \\ &-\frac{\partial \overline{p}}{\partial x} + \mu\left[\frac{\partial^2 \overline{u}}{\partial x^2} + \frac{\partial^2 \overline{u}}{\partial y^2} + \frac{\partial^2 \overline{u}}{\partial z^2}\right] - \left[\frac{\partial}{\partial x}(\rho\overline{u'^{\,2}}) + \frac{\partial}{\partial y}(\rho\overline{u'v'}) + \frac{\partial}{\partial z}(\rho\overline{u'w'})\right]\end{aligned}$$

to finally obtain:

$$\begin{aligned}\rho\left[\frac{\partial\overline{u}}{\partial t}+\frac{\partial}{\partial x}(\overline{u}^2)+\frac{\partial}{\partial y}(\overline{uv})+\frac{\partial}{\partial z}(\overline{uw})\right]=\\ -\frac{\partial\overline{p}}{\partial x}+\left[\frac{\partial}{\partial x}\left(\mu\frac{\partial\overline{u}}{\partial x}-\rho\overline{u'^{\,2}}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial\overline{u}}{\partial y}-\rho\overline{u'v'}\right)+\frac{\partial}{\partial z}\left(\mu\frac{\partial\overline{u}}{\partial z}-\rho\overline{u'w'}\right)\right]\end{aligned} \tag{2.33}$$

The formulation (2.33) is the standard representation of the Reynolds-averaged momentum balance equation.
As it can be seen there's a new term that appears due to the time-averaging of the Navier-Stokes momentum equation, this term is part of the total shear stress and represents the part of shear stress that is created due to the fluctuations of the turbulence. Since this additional term arises from the Reynolds averaging operation it has been called **Reynolds stress**. It's possible to make an analogy with the shear stresses that are present in the Navier-Stokes equations, the latter are due to the transport of momentum due to the random motion of molecules. Similarly, the Reynolds stresses are the effect of the convection of momentum due to the turbulent flow. Note that shear stresses are an effect with a molecular origin, while the shear stresses are due to the convective, fluctuating motion of fluid packets (the aforementioned eddies).
The time-averaged momentum equations for the other two directions, y and z, are obtained in the exact same way of the (2.33).
In general, it's possible to write the time-averaged momentum balance equation in the generic i-direction as:

$$\rho\frac{D\overline{u}_i}{Dt}=-\frac{\partial\overline{p}}{\partial x_i}+\mu\nabla^2\overline{u}_i-\rho\left(\frac{\partial\overline{u_i'u_j'}}{\partial x_j}\right) \tag{2.34}$$

with the rightmost term representing the **Reynolds stress tensor**.
The Reynolds stress tensor is a 3×3 symmetric tensor with a total of 6 unique components:

$$\rho\left(\overline{\mathbf{v'v'}}\right)=\rho\begin{bmatrix}\overline{u'^{\,2}} & \overline{u'v'} & \overline{u'w'}\\ \overline{v'u'} & \overline{v'^{\,2}} & \overline{v'w'}\\ \overline{w'u'} & \overline{w'v'} & \overline{w'^{\,2}}\end{bmatrix} \tag{2.35}$$

The first thing to note of the RANS equations system is that the time-averaging of the Navier-Stokes equations brings with it additional quantities:

- 4 mean values $\overline{p}$, $\overline{u}$, $\overline{v}$, $\overline{w}$
- 6 unique terms of the Reynolds stress tensor

for a total of 10 additional unknowns.
Since the equations remain always 4 (continuity equation and the 3 components

of the momentum balance equation), the RANS equation system is not a closed problem and, to be solved, it needs additional equations to model the components of the Reynolds stress tensor and finally close the problem. This problem led to the definition of the various turbulence models that are used today in modern CFD and which are the subject of the next section.

2.2 Turbulence Models

In order to solve the closure problem of the RANS equations system, it's necessary to find a way to evaluate the Reynolds stresses. Unfortunately, an exact theory that defines the Reynolds stresses doesn't exist, all the models that are used in modern CFD are based on semi-empirical assumptions.

Eddy Viscosity concept

This is the simplest model to approximate the Reynolds stresses, it's based on the concept of 'Eddy Viscosity' introduced by Joseph Valentin Boussinesq, in 1877 [11]. This method puts in relation the Reynolds stress tensor and the averaged-velocity gradients through the eddy viscosity μ_T which is a scalar, dimensional (I.S. $[Pa\ s]$) and isotropic quantity that can vary in space and time.
The 'Boussinesq relation' for the Reynolds stresses is:

$$-\overline{u_i' u_j'} = \mu_T \left(\frac{\partial \overline{u_i}}{\partial x_j} + \frac{\partial \overline{u_j}}{\partial x_i} \right) - \frac{2}{3} k \delta_{ij} \tag{2.36}$$

where:

- μ_T is the eddy viscosity.
- $k = \frac{1}{2}\overline{u_i' u_i'}$ is the Turbulence Kinetic Energy (TKE).
- δ_{ij} is the Kronecker's delta (previously mentioned in the paragraph 2.1, which is 1 if $i = j$ (normal stress) and 0 if $i \neq j$ (tangential stress).

In this model, the additional turbulence stresses are given by increasing the overall molecular viscosity through the eddy viscosity. There are several subcategories for the eddy viscosity models, depending on the number of (transport) equations solved to compute the eddy viscosity coefficient [17]:

- Zero equations (algebraic) models, for example the Prandtl's Mixing-Length model, Cebeci-Smith model, ...
- One equation models, for example the Spalart-Allmaras model, k-model, ...
- Two equations models, for example the k-ϵ model, k-ω model, k-τ model, ...

- Three equations models, such as the k-ϵ-A model

- Four equations models, such as the v^2-f model

The one equation Spalart-Allmaras model will be further investigated in the next paragraph as it's used as turbulence model in the CFD simulation subject of this thesis project.

The Spalart-Allmaras model

The Spalart-Allmaras (S-A) is a one equation model that introduces a modelled transport equation for the turbulent kinetic energy (TKE) [60].
In its standard formulation the S-A is a low-Reynolds model, which means that is used for solving the flow in the regions where the Re number is low and viscosity effects are relevant. For this reason it specifically requires that the regions near the wall, where the viscosity effects are present, are properly resolved (generally a parameter that represents the quality of the mesh near the wall is y^+, the normalized distance from the wall, and it has to be as close as possible to 1; later on this chapter there will be further discussions on the mesh and its features).
This method is particularly suitable for aerodynamics applications but it's not designed for generic industrial flows and leads to large errors (for example jet-like free shear flows) [22]. Compared to the low Reynolds number k-ϵ model, the Spalart-Allmaras model is generally considered more robust and is often used as a way to obtain a preliminary solution for more advanced models. It can give reasonable results on relatively coarse meshes for which the low Reynolds number k-ϵ model does not converge or even diverges [5].
The one-equation model solves a transport equation for a viscosity-like variable $\tilde{\nu}$ (also called 'Spalart–Allmaras variable'):

$$\begin{aligned}\frac{\partial \tilde{\nu}}{\partial t} + u_j \frac{\partial \tilde{\nu}}{\partial x_j} &= c_{b1}\left(1 - f_{t2}\right)\tilde{S}\tilde{\nu} - \left[c_{w1} f_w - \frac{c_{b1}}{k^2} f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 \\ &+ \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left(\left(\nu + \tilde{\nu}\right)\frac{\partial \tilde{\nu}}{\partial x_j}\right) + c_{b2}\frac{\partial \tilde{\nu}}{\partial x_j}\frac{\partial \tilde{\nu}}{\partial x_j}\right]\end{aligned} \tag{2.37}$$

where ρ is the density, $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity and μ is the dynamic viscosity.
Once the S-A variable is known, the eddy viscosity is given by the equation:

$$\mu_T = \rho \tilde{\nu} f_{v1} \tag{2.38}$$

with:

$$f_{v1} = \frac{\chi^3}{c_{v1}^3 + \chi^3} \qquad \text{where} \qquad \chi = \frac{\tilde{\nu}}{\nu} \tag{2.39}$$

The terms that appears in (2.37) are:

$$\tilde{S} = S + \frac{\tilde{\nu}}{k^2 d^2} f_{v2} \tag{2.40}$$

where $S = \sqrt{2\omega_{ij}\omega_{ij}}$ is the magnitude of the vorticity, $\Omega_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right)$ is the rotation tensor, d is the distance from the field point to the nearest wall and the coefficient f_{v2} is given by the equation:

$$f_{v2} = 1 - \frac{\chi}{1 + \chi f_{v1}} \tag{2.41}$$

The other terms in the S-A equation (2.37) are:

$$f_w = g\left[\frac{1 + c_{w3}^6}{g^6 + c_{w3}^6}\right]^{\frac{1}{6}} \tag{2.42}$$

where $g = r + c_{w2}\left(r^6 - r\right)$ and $r = min\left[\frac{\tilde{\nu}}{\tilde{S}k^2d^2},\ 10\right]$

And

$$f_{t2} = c_{t3}\, e^{-c_{t4}\chi^2} \tag{2.43}$$

The dimensionless constants are:

$$c_{b1} = 0.1355 \qquad c_{w2} = 0.3$$

$$c_{b2} = 0.622 \qquad c_{w3} = 2.0$$

$$\sigma = 2/3 \qquad c_{t3} = 1.2$$

$$k = 0.41 \qquad c_{t4} = 0.5$$

$$c_{v1} = 7.1 \qquad c_{w1} = 3.239$$

where c_{w1} is obtained from the others constant with the equation:

$$c_{w1} = \frac{c_{b1}}{k^2} + \frac{1 + c_{b2}}{\sigma}$$

The boundary conditions of the S-A model are defined near the wall and in the 'farfield', which is the free-stream external boundary of the physical domain:

$$\tilde{\nu}_{\text{wall}} = 0 \quad \text{and} \quad \tilde{\nu}_{\text{farfield}} = 3\nu_\infty \div 5\nu_\infty \tag{2.44}$$

where ν_∞ is the kinematic viscosity of the fluid in the farfield.
The formulation presented above, developed by P.R. Spalart and S.R. Allmaras in

1994 [60], is the most commonly used; since then, some modification have been implemented in order to improve the method. For example, to avoid numerical instability, the term $\tilde{S}$ in (2.40) should never go to zero or becoming negative, for this reason in 2012 [7] a new equation for $\tilde{S}$ was proposed.
Defining:

$$\overline{S} = \frac{\tilde{\nu}}{k^2 d^2} f_{v2}$$

The equation (2.40) becomes:

$$\tilde{S} = \begin{cases} S + \overline{S} & \text{if} \quad \overline{S} \geq -c_2 S \\ \\ S + \frac{S\left(c_2^2 S + c_3 \overline{S}\right)}{(c_3 - 2c_2)S - \overline{S}} & \text{if} \quad \overline{S} < -c_2 S \end{cases} \tag{2.45}$$

Where $c_2 = 0.7$ and $c_3 = 0.9$ [1].
Others modifications to this formulation led to several variants of the Spalart-Allmaras model, to name a few [42]:

- Negative Spalart-Allmaras One-Equation Model (SA-neg) [7]: developed to address issues related to under-resolved grids and non-physical transient states in discrete settings.

- Spalart-Allmaras One-Equation Model without the f_{t2} Term (SA-noft2) [9]: several implementations of the S-A model ignore the f_{t2} term, which is a numerical fix in the original model in order to make zero a stable solution to the equation.

- Spalart-Allmaras One-Equation Model with Rotation/Curvature Correction (SA-RC) [56]: variant of the original model developed to account for rotation and curvature effects.

2.3 Discretization of the Governing Equations

Considering the Navier-Stokes set of governing equations presented in section 2.1, these five equations (two scalar equations and a vector equation in 3 directions) together form a set of coupled, nonlinear partial differential equations. It is not possible to solve these equations analytically for most engineering problems. However, it is possible to obtain approximate computer-based solutions to the governing

[1] Note that, even with this modification, if the vorticity magnitude S is identically zero also the term $\tilde{S}$ becomes zero, in this conditions is necessary to impose the value of r, that appears in (2.42), to avoid dividing by zero. So, whenever $\tilde{S} = 0$, set $r = 10$.

equations for a variety of engineering problems, this is the subject matter of Computational Fluid Dynamics [50].
Broadly speaking, the strategy of CFD is to replace the continuous problem domain with a discrete domain using a grid. In the continuous domain, each flow variable is defined at every point in the domain, for instance, the pressure p in the continuous 1D domain would be given as:

$$p = p(x), \quad 0 < x < 1 \tag{2.46}$$

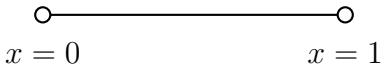

Figure 2.5. Continuous domain

In the discrete domain, each flow variable is defined only at the grid points.
So, in the discrete domain shown below, the pressure would be defined only at the N grid points:

$$p_i = p(x_i), \quad i = 1,2,\ldots,i,\ldots,N \tag{2.47}$$

In a CFD solution, one would directly solve for the relevant flow variables only at

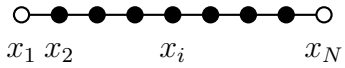

Figure 2.6. Discrete domain

the grid points, the values at other locations are determined by interpolating the values at the grid points.
The governing partial differential equations and boundary conditions are defined in terms of the continuous variables $p(x)$, $\vec{V}$ etc., as already said, they can be approximated in the discrete domain in terms of the discrete variables p_i, $\vec{V_i}$ etc. [50]. In this way the discrete system is a large set of coupled, algebraic equations in the discrete variables.
There are mainly three methods to discretize the computational domain and the set of equations:

- **Finite-Difference Method (FDM)**: A 2D computational domain is usually divided into hexahedral elements and the numerical solution is obtained at each node. The results is a system of discrete equations of the variables in each grid point.
  For spatial derivatives approximation there are different schemes that can be

adopted, according to the desired order of accuracy: Forward (or Backward) difference scheme (first order of accuracy), Central difference scheme (second order of accuracy, but intrinsically unstable), Upwind scheme (which can be both first order or second order accurate), etc.
FDM works well with regular grids, for non-uniform grids it's very difficult to use, for this reasons is rarely adopted for CFD solvers [46].

- **Finite-Volume Method (FVM)**: This method is based on the integrated version of the governing equations.
The 3D or 2D domain is discretized into finite volumes or cells, respectively, and the governing equations are solved for every volume/cell. The final form of the equations after discretization involves the calculation of the fluxes of the conserved quantities (mass, momentum and energy), at the volumes interfaces which is an essential part in this method.
Unlike FDM, finite volume approach is also suitable for non-uniform grids and, most important it is an intrinsically conservative[2] method (both FDM and FEM need to be carefully formulated to ensure the conservative solution). For this reasons the Finite-Volume approach is the most used discretization method for CFD solvers [46].

- **Finite-Element Method (FEM)**: This method is used in structural analysis of solids, but is also applicable to fluids. The main drawback of the FEM formulation is that it requires special care to ensure a conservative solution. It is much more stable than the finite volume approach but FEM can require more memory and has slower solution times than the FVM [68].

If the subject of the CFD simulation is time dependant there's also the necessity of a method for the time discretization.
There are two possible schemes:

- **Explicit scheme**: With this method the value of the conserved quantities (for example the pressure p) at the time-step t is only function of its value at the time-step $t-1$:
$$p^t = f(p^{t-1}) \tag{2.48}$$
This is the simplest scheme to implement in a CFD simulation because it doesn't need any kind of matrix inversion to update the conserved quantities

[2]Granting the 'conservative' formulation means that once the equations are discretized, the flux terms form a 'telescoping-series', that is when fluxes are summed into and out of a row of cells, the inter-cell fluxes cancel and the net flux is just the difference between the fluxes at the boundaries.
The main advantage of this formulation is that it allows discontinuous solution, which is necessary for shock-related problems.

at the next time-step. The downside of this method is that it's stable only if the Courant-Friedrichs-Lewy (CFL) condition is satisfied:

$$C = a\frac{\Delta t}{\Delta x} < 1 \tag{2.49}$$

where C is the Courant number and a is the propagation speed of the perturbation.

- **Implicit scheme**: with this method the updated value of the conserved quantities is function of its value at both the time-steps t and $t-1$:

$$p^t = f(p^t, p^{t-1}) \tag{2.50}$$

In this case, we can't update the variable at each grid point independently, we instead need to solve a system of algebraic equations in order to calculate the values at all grid points simultaneously. In any case, the Implicit scheme is not unconditionally stable for governing equations (due to their non-linear behaviour) but it's less limited regarding the Courant number adopted, which can be greater than 1.

Courant number is a key factor in CFD simulations as it defines the time-step with which the simulation advances. Taking larger time-steps leads to faster convergence to the steady state, so it is advantageous to set the Courant number as large as possible, within the limits of stability.

The computational Grid

Creating a high-quality computational grid (or 'mesh') is one of the most critical factors that must be considered to ensure simulation accuracy. A mesh partitions the space of the domain into elements (or 'cells') over which the equations can be solved, which then approximates the solution over the complete domain [24].
The 2 major aspects that characterize a mesh are the element shape and the grid structure.

Element Shape

For a 2-dimensional grid the most common cell shapes are: **Triangular**, which is the simplest one and the most common used for 'unstructured' grids, and **Quadrilateral**, most commonly used for 'structured' grids.
If the domain is 3-dimensional the elements can have different shapes depending on the complexity of the geometry. If the 3D domain is obtained from an extrusion of a 2D model the grid can be constituted by **Triangular prisms** or **Hexaedron** obtained by direct extrusion of the triangular o quadrilateral cells from the 2D model.

Other common element shapes for 3D meshes are the **Pyramid** (commonly used as transition elements between square and triangular faced elements and other in hybrid meshes and grids [24]), the **Tetrahedron** and **Polyhedron**, which is the most complex one as it can have any number of faces depending on the geometry and requires more computational effort per cell compared to others shapes.

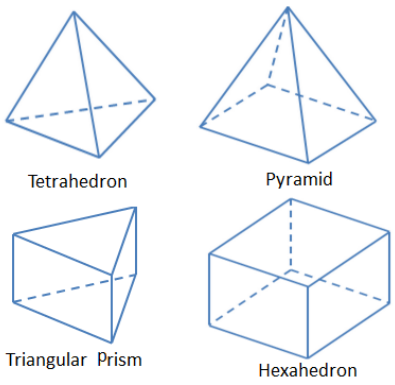

Figure 2.7. 3D cell shapes representation [24]

Grid Structure

There are three types of grid structure:

- **Structured Grids**: they are characterized by regular connectivity between the elements of the grid.It has a high spatial efficiency and leads to a better resolution of the domain and a faster convergence but it's limited to relatively simple and regular geometries.

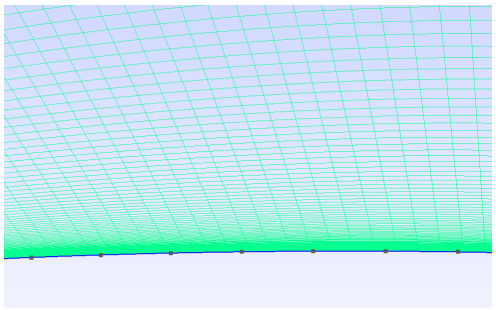

Figure 2.8. Structured mesh (software: Gmsh)

- **Unstructured Grids**: they are characterized by irregular connectivity between the elements. This is the simplest structure for a mesh but it gives more numerical diffusion than a structured mesh [71]. This means that a solution obtained on an unstructured mesh is more 'damped' compared to the solution obtained on a structured mesh. Numerical diffusion is not always an undesirable effect because it can increase the 'robustness' of the solution, leading to an easier convergence.

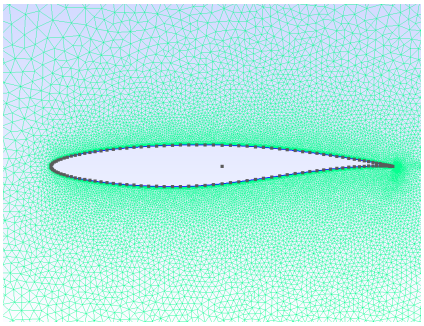

Figure 2.9. Unstructured mesh (software: Gmsh)

- **Hybrid Grids**: Hybrid grids integrate both structured and unstructured grids in order to improve the spatial efficiency of the overall mesh. They are particularly suitable for large domains with different complex geometries, the structured grid is used where the shape of the geometry is regular and the unstructured one is used to discretize the parts where the geometry is complex.
  A different application for hybrid meshes could be, for example, the discretization of a two-dimensional domain that contains an airfoil: if the simulation that needs to be defined involves molecular viscosity, the mesh in the near-wall region on the airfoil must have a high resolution (especially in the direction normal to the wall) in order to fully capture the high gradient of velocity that is characteristic of this region, which is called 'Boundary Layer'. For this reason to discretize boundary layers is common to use structured mesh with elements very thin near the wall which gradually become thicker moving away in the normal direction. Outside the boundary layer is possible to use a simple unstructured mesh with the elements that are small in the region near the airfoil where there are the major changes in the fluid properties, becoming larger and larger moving far from the airfoil in the region where the fluid is almost still.

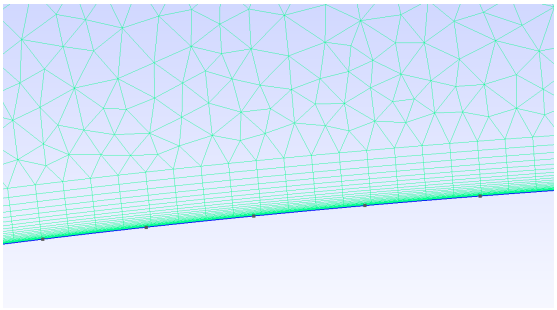

Figure 2.10. Triangular unstructured mesh outside the structured boundary layer's mesh (software: Gmsh)

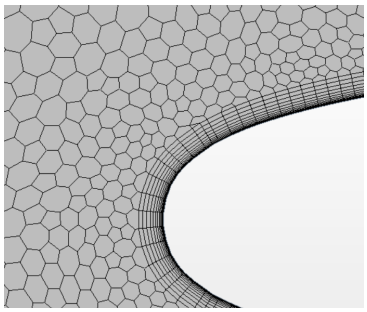

Figure 2.11. Hexagonal unstructured mesh outside the structured boundary layer's mesh (software: Siemens Simcenter STAR-CCM+ v.2019.3)

Mesh Quality

The mesh quality determines the accuracy of the solution and the convergence velocity, these two aspects are inversely proportional: refining the mesh, reducing the elements size, leads to an increase in accuracy but brings with it an increase in computational cost of the simulation.
The three aspects that a mesh needs to ensure to the solution are:

- **Accuracy**, which means ensuring that the discretization error[3] is the lowest possible.
- **Rate of convergence**, after every iteration of the simulation the truncation error[4] needs to be as low as possible, leading to a faster convergence.
- **Grid independence**, a solution is considered grid-independent if the discretization and truncation error are small enough given sufficient iterations, this aspect is important for comparative results [24].

There are different parameters used to measure the quality of the elements that form the grid, and they can vary from software to software. Taking as example a 2-dimensional grid constitute by triangular elements, some common parameters used to measure the grid quality are:

[3]The discretization error is the error due to the approximation of the derivatives with a finite difference:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \simeq \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

[4]The truncation error is the difference between the exact solution and the approximated solution obtained from the discretization

- **Skewness**. The skewness is a suitable indicator of the overall quality of the mesh, for triangular elements, with reference to Fig. 2.12, the skewness is given by the equation:

$$\text{skewness} = 1 - \frac{\overline{e\,e'}}{\sqrt{A}} \tag{2.51}$$

where e is the inner face center, e' is the center of $\overline{PE}$ and A is the area of the inner face. The skewness indicates the distance between the connecting

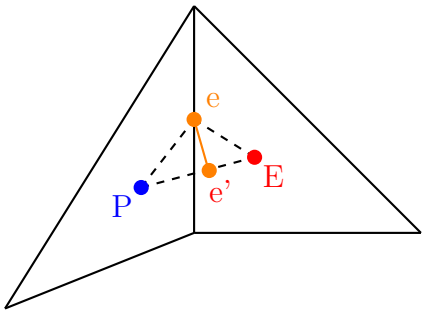

Figure 2.12. Definition of skewness

center and face center, if these two centers, e and e', are coincident, the skewness is equal to 1.

- **Aspect Ratio** (γ). For a triangular element the aspect ratio is given by the ratio between the diameter of the inscribed circle and the radius of the circumscribed circle of the triangle:

$$\gamma = \frac{2r_i}{r_e} \tag{2.52}$$

The value of γ needs to be as close as possible to 1 to ensure the regularity of the elements of the grid.

- **Signed Inverse Condition Number (SICN)**. Is a parameter based on the definition of the condition number of the transformation matrix S between the ideally shaped element and the actual element.

- **Normalized wall distance** (y^+). This is a parameter that measures the quality of the mesh of the boundary layer in near-wall regions of the domain. It depends on the flow conditions (cinematic viscosity ν, density ρ and dynamic viscosity μ), the wall shear stress τ_w and the normal distance of the

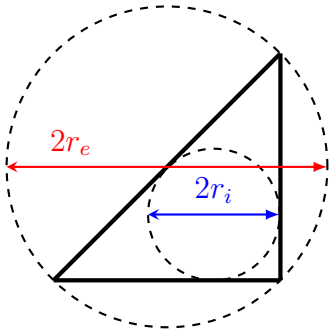

Figure 2.13. Inscribed and circumscribed circles in a triangular element

first cell center from the wall Δy:

$$y^+ = \frac{\Delta y}{\nu}\sqrt{\frac{\tau_w}{\rho}} \qquad \text{with} \quad \tau_w = \mu\left(\frac{\partial u}{\partial y}\right)_{y=0} \tag{2.53}$$

The ideal value of y^+ for a turbulent boundary layer modelled with the Spalart-Allmaras model (see paragraph 2.2) is $y^+ = 1$, this is not a universal rule and it depends of the turbulence model used and the associated wall functions (if present). Due to this fact, the different models require different values of y^+ as well as different spatial resolutions of the near wall area. Note that differently from the previous quality indicators of the mesh, which are strictly related to the geometry of the elements, the y^+ can't be known before the actual simulation of the flow because it's related to the flow quantities that need to be computed.

Discretization Scheme

Considering a generic conservation law problem, it's integral form over a control-volume V with external surface S is:

$$\int_V \frac{\partial \mathbf{W}}{\partial t}\, dV + \int_S \mathbf{F}(\mathbf{W}) \cdot \mathbf{n}\, dS = 0 \tag{2.54}$$

Where $\mathbf{W}$ is the vector of the conserved quantities and $\mathbf{F}(\mathbf{W})$ is the corresponding flux tensor [39]. The semi-discrete formulation of this system of equation with the finite volume method, for the generic i cell, is:

$$\frac{\partial}{\partial t}\mathbf{W}_i \Delta V_i + \sum_{S_j} \left(\mathbf{F}(\mathbf{W}) \cdot \mathbf{n}\right)_j \Delta S_j = 0 \tag{2.55}$$

where $\mathbf{W}_i$ is the cell-averaged value of the conserved quantities, ΔV_i is the cell volume, ΔS_j is the j-th face surface of the cell.

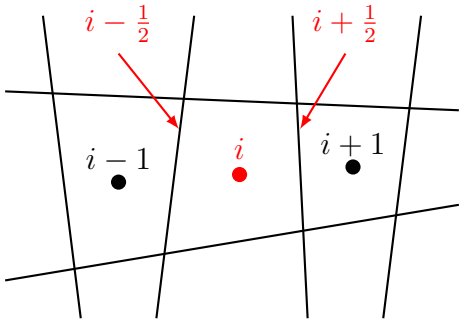

Figure 2.14. 2-dimensional cell, interfaces indices $i \pm \frac{1}{2}$

The second term of the formulation in (2.55) collect the fluxes of the conserved quantities evaluated at the interface between the cells and they can be evaluated by interpolation of the values of the conserved quantities at the cell centers. In the governing equations for a viscous fluid, there are two kinds of fluxes: the convective fluxes due to the convective transport of the conserved quantities and the diffusive fluxes due to the viscous transport of the conserved quantities. The convective fluxes are the most critical ones because, in general, the viscosity of the flow acts as 'natural' stabilizer of the diffusive term, on the other hand the convective term isn't stabilized by its own, so particular care needs to be used when determining this term.

Convective Schemes

Since the focus of the definition of the discretization scheme is only the convective term, in order to relieve the notation, in this section will be considered the Euler equations[5] for a one-dimensional domain. In general, the Euler equations in 3-dimensions can be expressed in a compact way as:

$$\frac{\partial}{\partial t}\int_V \mathbf{W}\, dV + \int_S \mathbf{F}_c \cdot \mathbf{n}\, dS = 0 \tag{2.56}$$

$$\text{Where} \quad \mathbf{W} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{Bmatrix} \quad \text{and} \quad \mathbf{F}_c = \begin{Bmatrix} \rho\mathbf{v} \\ p\mathbf{i} + \rho u\mathbf{v} \\ p\mathbf{j} + \rho v\mathbf{v} \\ p\mathbf{k} + \rho w\mathbf{v} \\ (E+p)\mathbf{v} \end{Bmatrix}$$

[5]The Euler equations are governing equation obtained from the Navier-Stokes equations neglecting the viscosity and the thermal conductivity of the flow.

with ρ the fluid density, p is the pressure, E is the total energy of the fluid and $\mathbf{v}$ the flow velocity vector with components u, v and w. Simplifying to the 1D case in the x-direction, equation (2.57) is obtained:

$$\frac{\partial}{\partial t}\int_{x_1}^{x_2} \mathbf{W}\, dx + \mathbf{F}_c(x_2) - \mathbf{F}_c(x_1) = 0 \tag{2.57}$$

$$\text{Where} \quad \mathbf{W} = \begin{Bmatrix} \rho \\ \rho u \\ E \end{Bmatrix} \quad \text{and} \quad \mathbf{F}_c = \begin{Bmatrix} \rho \mathbf{v} \\ p + \rho u^2 \\ (E+p)u \end{Bmatrix}$$

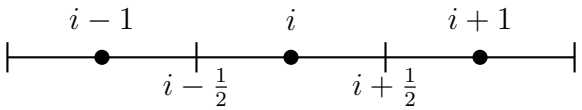

Figure 2.15. 1D domain discretized with FVM

The Euler equations written for the i cell of the discretized domain in fig. 2.15 are:

$$\frac{\partial \mathbf{W}_i}{\partial t}\Delta x + \mathbf{F}_c(x_{i+\frac{1}{2}}) - \mathbf{F}_c(x_{i-\frac{1}{2}}) = 0 \tag{2.58}$$

where $\mathbf{W}_i$ is the vector of the conserved quantities evaluated in the cell center and $\mathbf{F}_c(x_{i+\frac{1}{2}})$ and $\mathbf{F}_c(x_{i-\frac{1}{2}})$ are the convective fluxes evaluated at the cell interfaces. Finally, since $\mathbf{F}_c = \mathbf{F}_c(\mathbf{W})$, the fluxes can be written as:

$$\frac{\partial \mathbf{W}_i}{\partial t}\Delta x + \mathbf{F}_c(\mathbf{W}_{i+\frac{1}{2}}) - \mathbf{F}_c(\mathbf{W}_{i-\frac{1}{2}}) = 0 \tag{2.59}$$

With this formulation is possible to define the convective fluxes at the cell interface using the values of the conserved quantities at the cell center, which are known. The basic discretization schemes that can be used to evaluate the convective fluxes are:

- **Central Difference Scheme (CDS)**: The most obvious way to define the fluxes at the interface is a linear interpolation:

$$\mathbf{F}_c(\mathbf{W}_{i+\frac{1}{2}}) = \mathbf{F}_c\left[\frac{1}{2}\left(\mathbf{W}_{i+1} + \mathbf{W}_i\right)\right] \tag{2.60}$$

  This scheme has a second order accuracy[6] in space but it's intrinsically unstable for flows characterized by strong convection or flow discontinuities like

[6]The order of accuracy of a method represents the rate of convergence discussed in paragraph 2.3, given an approximate solution $u_{\Delta x}$, a method is n-th order accurate if the error $E(\Delta x)$ is proportional to:

$$E(\Delta x) = ||u - u_{\Delta x}|| \propto \Delta x^n$$

Where Δx is the spacing of the grid.

shock waves. It can be used for very low Reynolds number flow where the diffusive term dominates over the convection [16].

- **Upwind Difference Scheme (UDS)**: Also known as First-Order Upwind scheme (FOU scheme), is based on the concept of 'taking the information from the direction from which the flow is coming', it's mathematically defined as:

$$\mathbf{F}_c(\mathbf{W}_{i+\frac{1}{2}}) = \begin{cases} \mathbf{F}_c(\mathbf{W}_i) & \text{if} \quad \mathbf{v}\cdot\mathbf{n} > 0 \\ \\ \mathbf{F}_c(\mathbf{W}_{i+1}) & \text{if} \quad \mathbf{v}\cdot\mathbf{n} < 0 \end{cases} \tag{2.61}$$

  If the velocity vector is directed rightward, a backward scheme is used, otherwise, a forward scheme is used.
  As mentioned this is a first order accurate scheme (because both backward and forward schemes are first order accurate) so it's less accurate than the CDS but it's highly stable. The downside of this scheme is that it's highly diffusive, specially when the flow direction is skewed relative to the grid lines [16].

- **Hybrid Difference Scheme (HDS)**: Defined by Dudley Brian Spalding in 1970 [61], the HDS is a numerical scheme that combines the aforementioned CDS and UDS. The method consists of using the second order CDS only for the cells where the local Pèclet[7] number is sufficiently low ($Pe < 2$), otherwise the first order Upwind scheme will be used.

In order to obtain higher accuracy order methods and ensure the stability of the solution, even in high Reynolds number flows where the convective effect is dominant, more advanced discretization schemes were developed. Some of these methods are based on the second order CDS combined with a scalar artificial diffusion term to increase the stability (Lax-Friedrichs method or Jameson-Schmidt-Turkel scheme). Another concept on which several numerical methods are based is the REA Algorithm, where the acronym stands for: Reconstruct, Evolve and Average, which are the basic steps of the algorithm:

1) The solution is reconstructed using a polynomial piece-wise function (constant piece-wise functions gives first order accuracy, linear piece-wise functions gives a second accuracy, etc.)

[7]The Pèclet number is the numerical counterpart of the Reynolds number, it measures the relative strengths of diffusion and convection and it's defined as:

$$Pe = \frac{\rho u \Delta x}{\Gamma}$$

Where ρ is the density, u is the velocity, Γ is the numerical diffusion coefficient and Δx the cell dimension.

2) The reconstructed solution is then evolved at the successive time step, introducing discontinuities in the cells interfaces which have to be solved using the Riemann problems theory[8].

3) The evolved reconstructed solution is then averaged to obtain the new cell averaged values.

Higher order methods of this family also introduce slope limiting function to prevent oscillations and unstable behaviours (Minmod limiter, Van Albada limiter, Venkatakrishnan limiter, etc.). Some numerical schemes that belongs to this family are: Monotonic Upwind Scheme for Conservation Laws (MUSCL), Essentially Non-Oscillatory (ENO) schemes, Harmonic Quadratic Upwind Interpolation Convective Kinetics (H-QUICK).
In this section will be further discussed the **Jameson-Schmidt-Turkel (JST)** central scheme as it's the one used for the CFD simulation object of this thesis project. As mentioned, the JST scheme is based on the CDS but with an additional artificial dissipative flux term. The convective flux evaluated at the cell interface with the JST scheme is given by the equation:

$$\mathbf{F}_c(\mathbf{W}_{i+\frac{1}{2}}) = \mathbf{F}_c\left[\frac{1}{2}\left(\mathbf{W}_{i+1} + \mathbf{W}_i\right)\right] - \mathbf{D}_{i+\frac{1}{2}} \tag{2.62}$$

where $\mathbf{D}_{i+\frac{1}{2}}$ is the diffusive flux term given by:

$$\mathbf{D}_{i+\frac{1}{2}} = \epsilon^{(2)}_{i+\frac{1}{2}}\Delta\mathbf{W}_{i+\frac{1}{2}} - \epsilon^{(4)}_{i+\frac{1}{2}}\left(\Delta\mathbf{W}_{i+\frac{3}{2}} + 2\Delta\mathbf{W}_{i+\frac{1}{2}} + \Delta\mathbf{W}_{i-\frac{1}{2}}\right) \tag{2.63}$$

and $\Delta\mathbf{W}_{i+\frac{1}{2}}$ is a vector with components:

$$\Delta\mathbf{W}_{i+\frac{1}{2}} = \left\{\begin{array}{c} \rho_{i+1} - \rho_i \\ (\rho u)_{i+1} - (\rho u)_i \\ E_{i+1} - E_i \end{array}\right\} \tag{2.64}$$

The spectral radius[9] of the Jacobian matrix for the i-th cell is $r_i = |u| + c$ where c is the local speed of sound [34].
The dissipative coefficients $\epsilon^{(2)}_{i+\frac{1}{2}}$ and $\epsilon^{(4)}_{i+\frac{1}{2}}$ are switched on and off by a 'pressure sensor':

$$s_i = \left|\frac{p_{i+1} - 2p_i + p_{i-1}}{p_{i+1} + 2p_i + p_{i-1}}\right| \tag{2.65}$$

[8]Named after Bernhard Riemann [64], solving a Riemann problem means to evaluate the evolution in time of a initial discontinuity.

[9]The spectral radius of a square matrix is the largest absolute value of its eigenvalues [29]

At the cell interfaces the spectral radius and the sensor are defined as:

$$r_{i+\frac{1}{2}} = max\left(r_{i+1},\ r_i\right) \qquad \text{and} \qquad s_{i+\frac{1}{2}} = max\left(s_{i+1},\ s_i\right)$$

It's now possible to evaluate the dissipative coefficients $\epsilon^{(2)}_{i+\frac{1}{2}}$ and $\epsilon^{(4)}_{i+\frac{1}{2}}$:

$$\epsilon^{(2)}_{i+\frac{1}{2}} = k_2 s_{i+\frac{1}{2}} r_{i+\frac{1}{2}} \tag{2.66}$$

$$\epsilon^{(4)}_{i+\frac{1}{2}} = max\left(0,\ k_4 r_{i+\frac{1}{2}} - c_4 \epsilon^{(2)}_{i+\frac{1}{2}}\right) \tag{2.67}$$

The constants in (2.66) and (2.67), for a transonic flow simulation, can be assumed to be equal to:

$$k_2 = 1 \qquad k_4 = \frac{1}{32} \qquad c_4 = 2$$

In any case, as a general rule, k_2 should be chosen to give enough lower order dissipation to prevent unstable oscillations in the vicinity of shock waves, while c_4 should be large enough to make sure that the higher order terms are turned off when the lower order dissipation is active [34].
Summarizing, the JST Scheme offers a good compromise between accuracy and robustness as the 2-nd and 4-th order dissipation coefficients can be turned off, by the pressure sensor, in presence of strong discontinuities (like shock-waves) that could introduce unwanted oscillatory behaviours. The main drawback of this method is that the additional artificial diffusion could lead to an over prediction of the viscous drag contributions in low Reynolds flows.

Chapter 3

The Multi-Layer Perceptron

This chapter contains an overview of the second generation of Artificial Neural Networks (ANNs), which are also called Multi-Layer Perceptrons (MLPs). In the next pages the architecture of this type of Neural Networks (NNs), their features, the algorithms on which they are based and the theory behind the training process are going to be discussed and analyzed.
The MultiLayer Perceptron is a supervised learning algorithm that learns a function (3.1), after training on a Dataset [53].

$$f(x) : \mathbb{R}^I \rightarrow \mathbb{R}^O \qquad \text{with } I = \text{Input dimension}\,,\ O = \text{Output dimension} \tag{3.1}$$

The database consists in samples of **Input features** $x = x_1,\ x_2,\ x_3, \cdots,\ x_I$ and corresponding **Targets** $y = y_1,\ y_2,\ y_3, \cdots,\ y_O$.

INPUT FEATURES	**TARGETS**
$[x_1,\ x_2,\ x_3, \cdots,\ x_I]_1$	$[y_1,\ y_2,\ y_3, \cdots,\ y_I]_1$
$[x_1,\ x_2,\ x_3, \cdots,\ x_I]_2$	$[y_1,\ y_2,\ y_3, \cdots,\ y_I]_2$
$\vdots$	$\vdots$
$[x_1,\ x_2,\ x_3, \cdots,\ x_I]_N$	$[y_1,\ y_2,\ y_3, \cdots,\ y_I]_N$

Table 3.1. Structure of a Database with N samples of input features and targets

Giving to the MLP a sample of input features and targets, it can learn an approximation of the function $f(x)$ by regression or classification.
As already mentioned in section 1.2, the basic unit of a neural network is the Artificial Neuron which is basically a mathematical function defined as a model of the biological neuron [8]. For the second generation of ANN the function behind the output of a k-th artificial neurons, which receives I input signals x_i is:

$$y_k = g\left(\sum_{i=0}^{I} w_{ki}x_i + b_k\right) \tag{3.2}$$

The term in (3.2) in brackets is called weighted input and it depends on the weights w that connects the two neurons, which can be considered similar to the coefficients of a linear regression, and on a constant signal b called 'bias'. The function $g(x) : \mathbb{R} \to \mathbb{R}$ that takes as arguments the weighted input is the activation function (or transfer function) which gives the final output, also called neuron activation, of the artificial neuron [21].

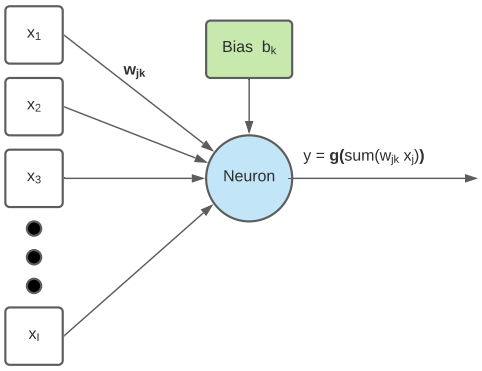

Figure 3.1. Artificial Neuron

The neurons are organized in layers, in a fully connected architecture every neuron of the layer receives the signal from every feature of the input and produces an output that becomes the input of the next layer. A common structure of a MLP consists of 3 layers: the **Input layer** which contains the input values, an **hidden layer** which has an arbitrary dimension, depending on the complexity of the function that has to be learned, and acts as intermediate elaboration layer, and the **Output layer** which contains the neurons that return the final output of the network. Considering, as example, the aforementioned MLP with 3 layers (Fig. 3.2):

1) Input layer with dimension I (index: i)

2) Hidden layer with dimension H (index: h)

3) Output layer with dimension O (index: j)

The sequence of the elaboration process from the input to the output ('Forward' step, or 'Inference'), for the j-th output neuron will be:

$$y_h = g_{\text{hid}} \left(\sum_{i=1}^{I} w_{hi} x_i + b_h \right) \qquad \text{Input} \to \text{Hidden} \tag{3.3}$$

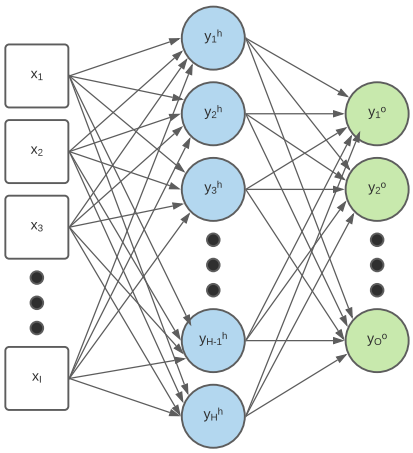

Figure 3.2. MLP architecture representation

$$y_j = g_{\text{out}} \left(\sum_{k=1}^{H} w_{jk} y_h + b_j \right) \qquad \text{Hidden} \rightarrow \text{Output} \tag{3.4}$$

Putting together the equations (3.3) and (3.4) the overall sequence is:

$$y_j = g_{\text{out}} \left[\sum_{k=0}^{H} w_{jk}\, g_{\text{hid}} \left(\sum_{i=0}^{I} w_{ki} x_i + b_k \right) + b_j \right] \qquad \text{Input} \rightarrow \text{Output} \tag{3.5}$$

It's easy to observe that the activation of a neuron is a function of the activation of the neuron in the previous layer.
In the Multi-Layer Perceptron the hidden layers can be more than one, increasing the depth of the architecture and leading to the capability to learn more complex functions. For a MLP with L layers, the equation (3.5) can be expressed as:

$$y_j = g^L \left(\sum_{k_L=1}^{H_L} w_{jk}^L\, g^{L-1} \left(\sum_{k_{L-1}=1}^{H_{L-1}} w_{jk}^{L-1} \cdots g^1 \left(\sum_{i=1}^{I} w_{ki}^1\, x_i + b_k^1 \right) \cdots + b_k^{L-1} \right) + b_k^L \right) \tag{3.6}$$

where w_{jk}^l is the weight that connects the k-th neuron of the $(l-1)$-th layer to the j-th neuron of the l-th layer.
To relief the notations is possible to write the previous equation using a vector formulation [44]. We indicate with $\mathbf{a}^l$ the **activation vector** of the neurons inside the l-th layer:

$$\mathbf{a}^l = g^l \left(\mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \right) \tag{3.7}$$

where $\mathbf{w}^l \in \mathbb{R}^{j \times k}$ is the **weight matrix** for the l-th layer, the components of this matrix are the weight connections w_{jk}^l between the k-th neurons of the layer and the

the j-th neurons of the successive layer, and $\mathbf{b}^l$ which is the **bias vector** containing all the bias b_j^l of the considered layer.
It's also worth treating the weighted input as a standalone term [44], calling $\mathbf{z}^l$ the **weighted input vector** containing the weighted input of the neurons in the layer l and defined as:

$$\mathbf{z}^l = \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l \tag{3.8}$$

Using this vector notation it's possible to write the more compact version for the (3.6):

$$\mathbf{a}^L = g^L \left(\mathbf{w}^L \, g^{L-1} \left(\mathbf{w}^{L-1} \, \cdots \, g^1 \left(\mathbf{w}^1 \mathbf{x}_{\text{in}} + \mathbf{b}^1 \right) \, \cdots \, \mathbf{b}^{L-1} \right) \mathbf{b}^L \right) \tag{3.9}$$

3.1 Activation function

As mentioned at the beginning of this chapter, the activation function $g(x)$ is a function that receives as input the weighted input z_j^l of the j-th neuron in the l-th layer and returns the activation state of the neuron a_j^l. The task of the activation function is to introduce non-linearity inside the neural network, without the activation function the activation of the neuron is decided only by the weighted input, which is a linear function of the input set x_j (3.8). In this case the neural network can only perform linear regression task, even with a network made of several hidden layers, the output activation $\mathbf{a}^L$ will still be a linear function of the input layer $\mathbf{x}_{in}$ [63]:

$$\mathbf{a}^L = \mathbf{w}^L \left(\mathbf{w}^{L-1} \, \cdots \, \left(\mathbf{w}^1 \mathbf{x}_{in} + \mathbf{b}^1 \right) \, \cdots \, + \mathbf{b}^{L-1} \right) + \mathbf{b}^{L-1} \tag{3.10}$$

This happens because the inference is a linear combination of the neuron activation and a linear combination of linear functions is still a linear function. This means that any additional hidden layer is useless and the network can be replaced by a single layer network that achieves the same task [54].
Enabling non-linear regression is not the only task of the activation function, without it if we look at the error in the neurons of the output layer δ_j^L, computed with eq. (3.21):

$$\delta_j^L = \frac{\partial C_i}{\partial a_j^L} \, g' \left(z_j^L \right)$$

the derivative of the activation function (which in this case is a linear function) is a constant, this means that the gradient of the cost function is independent from the input and it's also constant. If there is an error in prediction, the changes made by backpropagation is constant and doesn't depend on the change in input [54].
The choice of a certain activation function mainly depends on the task that the NN has to perform, some the most common activation functions used in machine learning are [55]:

- **Sigmoid function**: also called Logistic function, is a monotonic differentiable function:

$$f(x)_{\text{sigmoid}} = \frac{1}{1+e^{-x}} \qquad f'(x)_{\text{sigmoid}} = f(x)\,[1-f(x)] \tag{3.11}$$

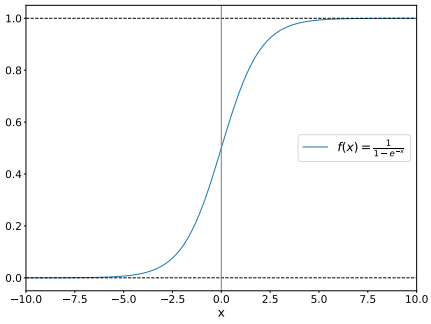

Figure 3.3. Sigmoid activation function

One of the best features of this function as activation function is that its image is [0, 1], which makes it particularly suitable for networks that predict probability [55]. Another important feature is that is very steep in the range $x \in [-2, 2]$ and then flattens out outside this range, this means that this function tends to return only values close to 0 or 1 making a clear distinctions in the output, which is a key feature for binary classifier networks [54].
The main drawback of the sigmoid activation function is that it's almost 'flat' at the tail ends of the curve, this mean that the gradient associated to this areas is very low, this is called 'Vanishing gradient' phenomena. If the activation of the neuron falls in this range of values the gradients of the cost function (which is related to the gradient of the activation function) will be very low and the weights and biases won't be further updated and basically the network will stop learning (or becomes very slow) [54].

- **Hyperbolic tangent function**:

$$f(x)_{\tanh} = \tanh x = \frac{2}{1+e^{-2x}} - 1 \qquad f'(x)_{\tanh} = 1 - f(x)^2 \tag{3.12}$$

The Hyperbolic tangent function is a rescaled sigmoid function, the activation range goes from -1 to 1 and, with respect to the sigmoid, has a steeper gradient in the range $x \in [-2, 2]$ but presents the same 'Vanishing gradient' problem. The Hyperbolic tangent activation function is mainly used for binary classification tasks.

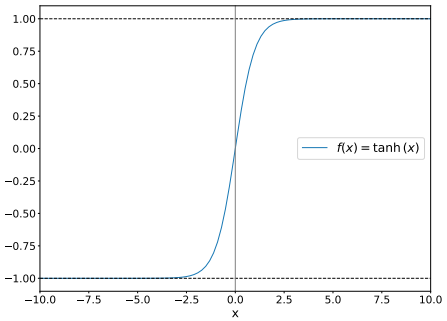

Figure 3.4. Hyperbolic tangent activation function

- **Rectified Linear Unit (ReLU)**: ReLU is the most widely used activation function for Machine Learning, it is defined as:

$$f(x)_{\text{ReLU}} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{3.13}$$

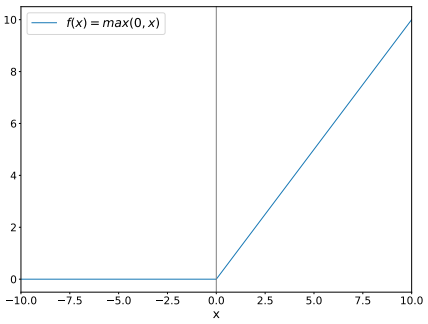

Figure 3.5. ReLU activation function

The ReLU function is non-linear and it's particularly suitable for regression models, every function can be approximated with a combination of ReLU-activated neurons [54]. The main feature that makes this activation function so popular is the fact that negative inputs in the neuron are returned as a zero by the ReLU, this cause a sparse activation[1] of the neurons in the network,

[1]Sigmoid causes a dense activation which means that all the neurons activate in the almost same way, which is computationally costly [54]

so that only a part of the neurons are active during the inference step, this makes this activation function very efficient.
This feature of the ReLU has a major downside: setting to zero the negative inputs causes the gradient of the cost function to go to zero instantly, this means that the neurons that go on this state don't respond anymore to variations in error or input. This issue is called 'dying ReLU', where several neurons in the network acts as a passive part of the NN.
To fix this issue is possible to use a variation of the ReLU activation function called Leaky ReLU where the negative part of the function is no more horizontal but has a linear behaviour with a very low slope:

$$f(x)_{\text{Leaky ReLU}} = \begin{cases} ax & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{3.14}$$

where a is the slope of the negative part and has to be sufficiently low to guarantee the sparse activation feature of the ReLU (the default value is often $a = 0.01$).

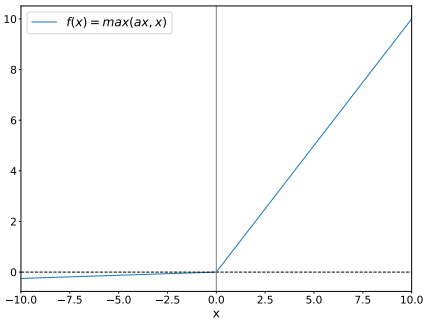

Figure 3.6. Leaky ReLU activation function

3.2 The Learning Process: Backpropagation Algorithm

Since now the focus was on the 'Forward' step of a neural network, from the input to the approximation of the output of a function, but nothing has been said on how the neural network learns how to properly approximate this output. The algorithm that allows the ANNs to learn the pattern behind the relations between Inputs and Outputs is called **Backpropagation algorithm**.
The concept behind the Backpropagation algorithm was first introduced in 1970, but its relevance was noted only in 1986, after the publication of the paper [52]. In the paper was described how the backpropagation algorithm was far faster than the previous learning algorithm used, making it possible to use neural nets to solve problems which had previously been insoluble [44]. The focus of this algorithm is to compute the partial derivative of the 'Cost function' (or 'Loss function') $C(w)$ with respect of the weights and the biases of the neural network:

$$\frac{\partial C(w)}{\partial w} \quad \text{and} \quad \frac{\partial C(w)}{\partial b}$$

This mathematical expression links how the cost changes when the weights and the biases change in the network.
The cost function used depends on the application of the neural network. For example, for classification tasks the cost function is usually a Cross-Entropy loss function defined as [18]:

$$C_{\text{CE}} = \frac{1}{N} \sum_{\text{class}}^{N} \left(-\log \left[\frac{\exp\left(x_{\text{class}}\right)}{\sum_j \exp\left(x_j\right)} \right] \right) \tag{3.15}$$

Where x_{class} is the output with the target class index which is normalized with respect to the sum of the outputs, the results of the logarithm of this fraction is then averaged over the number of classes N.
If the task is a regression of a function $y = f(x)$ a suitable cost function is the Mean Squared Error [19]:

$$C_{\text{MSE}} = \frac{1}{N} \sum_{i}^{N} \left(\mathbf{a}^L - \mathbf{y}(\mathbf{x})\right)^2 \tag{3.16}$$

Where $\mathbf{a}^L$ is the activation of the output layer and $\mathbf{y}(\mathbf{x})$ is the target, N is the number of training sets of the database.
In general, the overall cost function can be seen as the average of all the cost functions of the single training sample C_i, with:

$$C_i = \left(\mathbf{a}^L - \mathbf{y}(\mathbf{x})\right)^2 \tag{3.17}$$

This observation is important because the backpropagation algorithm allows to evaluate the derivative of the cost function of the single training sample with respect to the weights and biases, the overall derivative of the cost function is then obtained averaging over all the training sets [44].
Another useful observation is that the cost function is a function of the activation of the last layer (output activation):

$$C_i = C_i\left(\mathbf{a}^L\right) \tag{3.18}$$

Note that, even if C_i is also function of the target output $\mathbf{y}(\mathbf{x})$, this dependency is irrelevant from the point of view of the derivative with respect to the weights and the biases because, if they change, only the output activation $\mathbf{a}^L$ changes.
Before deriving the expression of $\frac{\partial C_i}{\partial w}$ and $\frac{\partial C_i}{\partial b}$ is necessary to introduce an intermediate quantity that represents the 'error' inside the neurons of the l-th layer [44]. This quantity is named $\boldsymbol{\delta}^l$ and it's a vector quantity where its components are defined as:

$$\delta_j^l = \frac{\partial C_i}{\partial z_j^l} \tag{3.19}$$

where z_j^l is the weighted input, eq. (3.8), of the j-th neuron inside the l-th layer. The reason why this quantity is representative of the neuron's error is because, if a small change Δz_j^l is introduced in the weighted input of the j-th neuron in the l-th layer, the final output of the neuron will be $g^l\left(z_j^l + \Delta z_j^l\right)$. This error will propagate through the successive layers and finally, affects the output activation, thus the cost function will change by a quantity:

$$\Delta C_i = \frac{\partial C_i}{\partial z_j^l}\Delta z_j^l = \delta_j^l \Delta z_j^l \tag{3.20}$$

The equations behind the backpropagation algorithm are basically four, which together allow to evaluate the gradient of the cost function ∇C, which has the derivatives of the cost function with respect to all the weights and all the biases as components [44].

1) **The equation for the error in the Output Layer**:

$$\delta_j^L = \frac{\partial C_i}{\partial a_j^L}\, g'\left(z_j^L\right) \tag{3.21}$$

The first term of the eq. (3.21) is the rate of change of the cost function with respect of the activation of the j-th neuron, the second term $g'\left(z_j^L\right)$ measures how the activation function changes at z_j^L. Note that $g'\left(z_j^L\right)$ depends on the particular cost function chosen for the neurons. The (3.21) can be expressed in a vector formulation as:

$$\boldsymbol{\delta}^L = \nabla_a C_i \odot g'\left(\mathbf{z}^L\right) \tag{3.22}$$

where the symbol $\odot$ stands for the Hadamard product[2] and $\nabla_a C_i$ is the gradient of the cost function with respect to the neurons activation, which is a vector with components $\frac{\partial C_i}{\partial a_j^L}$.

2) **The equation that relates the error $\boldsymbol{\delta}^l$ to the error in the next layer $\boldsymbol{\delta}^{l+1}$:**

$$\boldsymbol{\delta}^l = \left[\left(\mathbf{w}^{l+1}\right)^T \boldsymbol{\delta}^{l+1}\right] \odot g'\left(\mathbf{z}^L\right) \tag{3.23}$$

where $\left(\mathbf{w}^{l+1}\right)^T$ is the transposed weight matrix of the $(l+1)$-th layer.
This equation represents the backpropagation of the error from the $(l+1)$-th layer towards the l-th layer [44]. Using the (3.22) to compute the error in the output layer, with the (3.23) is then possible to determine the error of the neurons in every layer of the network.

3) **The equation for the rate of change of the cost function with respect to the biases**:

$$\frac{\partial C_i}{\partial b_j^l} \tag{3.24}$$

Since the relation between the cost function and the biases is not explicit, to determine the derivative it's possible to use the Chain-rule for the derivatives[3] [38]:

$$\frac{\partial C_i}{\partial b_j^l} = \frac{\partial C_i}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C_i}{\partial z_j^l}\frac{\partial\left(w_{jk}^l a_j^{l-1} + b_j^l\right)}{\partial b_j^l} = \frac{\partial C_i}{\partial z_j^l} \cdot 1 \tag{3.25}$$

where, as previously mentioned in (3.19), $\frac{\partial C_i}{\partial z_j^l}$ is the error δ_j^l. Thus, the rate of change of the error with respect to the biases is just the error of the neurons

[2]The hadamard product is the elementwise product between two matrices [33]. Given $A,\ B \in \mathbb{R}^{n\times m}$ two matrices with elements a_{ij} and b_{ij} the Hadamart product between A and B is:

$$C = A \odot B = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1m}b_{1m} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2m}b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}b_{n1} & a_{n2}b_{n2} & \cdots & a_{nm}b_{nm} \end{bmatrix}$$

[3]The Chain-rule is a mathematical formula used to compute the derivative of composite functions.
The Leibniz's formulation for this rule states that: given f a function of the dependant variable y, which is in turn a function of the dependant variable x, the derivative of f with respect to x can be computed as [14]:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}\frac{\partial y}{\partial x}$$

in the l-th layer of the network:

$$\frac{\partial C_i}{\partial b_j^l} = \delta_j^l \tag{3.26}$$

4) **The equation for the rate of change of the cost function with respect to the weights in the network**:

$$\frac{\partial C_i}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{3.27}$$

The equation (3.27) relates the changes of the cost function with respect to the weights with the activation of the neuron input of the weight w_{jk} and the error from the neuron output of w_{jk} [44], which are both quantities that can be evaluated with the available equations.

Known the basic equations behind the backpropagation of the error, the step-by-step process of the backpropagation algorithm is [44]:

(0) The input layer $\mathbf{x}$ corresponds to the activation $\mathbf{a}^1$.

(1) For each layer $l = 2,\ 3,\ ,\ \cdots,\ L-1,\ L$ the weighted input $\mathbf{z}^l$, eq. (3.8), and the activation vector $\mathbf{a}^l$, eq. (3.7), are computed (inference step).

(2) At the end of the inference is evaluated the error of the neurons in the output layer $\boldsymbol{\delta}^L$ with the eq. (3.22).

(3) The error is then backpropagated, for each $l = L-1,\ L-2,\ ,\ \cdots,\ 3,\ 2$, the quantity $\boldsymbol{\delta}^l$ is computed, eq. (3.23).

(4) The final step consists in the evaluation of the gradients of the cost function with respect to the weights, eq. (3.27) and the biases, eq. (3.26):

$$\frac{\partial C_i}{\partial w_{jk}^l} \quad \text{and} \quad \frac{\partial C_i}{\partial b_j^l}$$

By their own these two quantities don't affect the learning of the NN, they need to be coupled with a learning algorithm that changes the weights and the biases of the network according to the values of these derivatives, in order to decrease the cost function and obtain a more accurate output from the network.

Gradient Descent (GD) algorithm

The gradient descent training algorithm is based on the concept of reaching the minimum of the cost function by updating the weights and the biases of the network iteratively, by a Δw_{jk}^{l} and Δb_{j}^{l}, depending on the value of the gradient of the cost function, which represents the slope of the function [28]. In figure 3.7 the cost function C (z-direction) with respect to the weights and the biases (x and y directions) is represented, to achieve gradient descent the cost function needs to be a convex function [28].

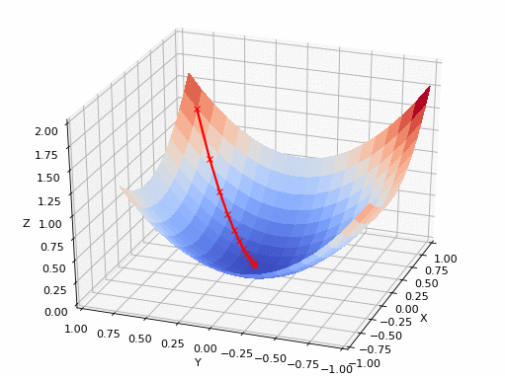

Figure 3.7. Cost function $C(w_{jk}^{l}, b_{j}^{l})$ representation [37]

After the backpropagation of the error, and the computation of the derivatives of the cost function, the weights and the biases of the neurons in the network are updated with the following equations:

$$\left(w_{jk}^{l}\right)^{e} = \left(w_{jk}^{l}\right)^{e-1} - \eta \left(\frac{\partial C}{\partial w_{jk}^{l}}\right)^{e-1} \qquad \left(b_{j}^{l}\right)^{e} = \left(b_{j}^{l}\right)^{e-1} - \eta \left(\frac{\partial C}{\partial b_{j}^{l}}\right)^{e-1} \tag{3.28}$$

where the apex e represents the iteration of the algorithm (also called 'Epoch'). The term η is the Learning Rate of the algorithm and it's a key factor for the training process, because it dictates how large are the 'steps' undertaken by the algorithm in the gradient descent. If the learning rate is too small, despite the algorithm will be more accurate in reaching the minimum of the cost function it will also be very slow. On the other hand, using a high learning rate leads to a faster convergence but this choice will makes the algorithm to converge to a sub-optimal solution or can even leads to a divergent behaviour (Fig. 3.8) as the algorithm begins to 'climb' the cost function instead of descending to the global minimum.
In order to always use the optimal learning rate is possible to define an 'adaptive' learning rate that starts with a high value at the beginning of the training (early epochs) when the algorithm is far away from the minimum, and once the loss stops

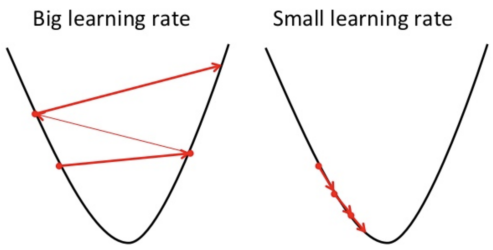

Figure 3.8. High learning rate VS Low learning rate [30]

decreasing it lowers to avoid the 'bouncing' of the algorithm and allowing it to reaches the global minimum.
There are different implementations for the gradient descent algorithm in neural networks applications, some of them are:

- **Batch Gradient Descent (BGD)** [28]: Also called 'Vanilla' Gradient Descent, is the basic implementation of this algorithm for a neural network. The cost function is evaluated for every training sample but only after the inference of every sample in the training database has been made the overall cost function is evaluated, the error backpropagated and the weights and biases are being updated (as previously mentioned, this is called 'training epoch'). The advantages of the BGD is that it's highly efficient and produce a smooth decrease of the cost function but it can also converge too early to a sub-optimal solution without reaching the actual global minimum of the cost function, resulting in a less accurate training.

- **Stochastic Gradient Descent (SGD)** [62]: with this method the update of weights and biases is made for every training sample of the database one-by-one.

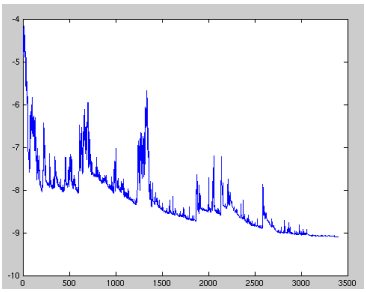

Figure 3.9. Fluctuation of the cost function with SGD algorithm [23]

This approach gives the SGD a higher rate of improvement in the decrease

of the cost function, on the other hand the high frequency of the updates is computationally more expensive and results in noisy gradients with the cost function that oscillates instead of steadily decrease (Fig. 3.9.

- **Adaptive Moment Estimation (Adam)** [36]: this is an improvement of the SGD algorithm that uses the first order moment (mean) and second order moments (variance) of the gradient to updates the weights and the biases.
  The algorithm is quite different from the previous ones, in the beginning, the first and second order moment vectors ($\mathbf{m}$ and $\mathbf{v}$) are initialized to 0, for every training iteration t they are updated following the equations:

$$\mathbf{m}^{t+1} = \beta_1 \mathbf{m}^t + (1 - \beta_1) \nabla_w C(\mathbf{w}) \tag{3.29}$$

$$\mathbf{v}^{t+1} = \beta_2 \mathbf{v}^t + (1 - \beta_2) \left(\nabla_w C(\mathbf{w})\right)^2 \tag{3.30}$$

  where β_1 and β_2 are called 'forgiving' factors and represent the exponential decay rates for the moment estimates, common values for these factors are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Note that in this case, in the notation of $\nabla_w C(\mathbf{w})$, the vector $\mathbf{w}$ contains all the internal parameters of the network that need to be updated, both the weights and the biases.
  The initialization of $\mathbf{m}$ and $\mathbf{v}$ to vectors of 0 leads to moment estimates that are biased towards zero, especially during the initial time steps [36], for this reason is necessary to introduce correction for this initialization bias:

$$\hat{\mathbf{m}}^{t+1} = \frac{\mathbf{m}^{t+1}}{1 - (\beta_1)^{t+1}} \qquad \hat{\mathbf{v}}^{t+1} = \frac{\mathbf{v}^{t+1}}{1 - (\beta_2)^{t+1}} \tag{3.31}$$

  where in this case $(\beta_1)^{t+1}$ and $(\beta_2)^{t+1}$ are the forgiving factors to the power $t + 1$. The new first and second order momentum obtained are called 'bias-corrected' momentum.
  The weights and biases can now be updated with the equation [36]:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\hat{\mathbf{m}}^{t+1}}{\sqrt{\hat{\mathbf{v}}^{t+1}} + \epsilon} \tag{3.32}$$

  where ϵ is a small scalar term (a common value assumed is $\epsilon = 10^{-8}$), used to prevent dividing by 0.

Note that often to increase the efficiency and the training velocity in the SGD and Adam algorithms the gradient of the cost function is not computed for every single training samples but after a 'mini-batch' (from now only called 'batch') of training samples, this is an important hyperparameter that can heavily influence the training process.

Chapter 4

Description of the Solution Techniques

In the first part of this chapter some details are presented and discussed related to the configuration and setup of the 2D airfoil transonic CFD simulations used to produce the Dataset on which the NNs were trained. The second part involves the analysis of the setup, features and architecture of the NNs developed in this project: first, the MLP that predicts the two aerodynamic coefficient CL (Lift Coefficient) and CD (Drag Coefficient) and then the MLP that predicts the flow field near the airfoil in terms of local Mach Number.

4.1 The CFD Simulations: Creating the Dataset

Before going into details with the configuration of the CFD simulations, it's worth briefly introducing the two main software used to run the simulations and creating the mesh.

The CFD Software

SU2

The software chosen for running the simulations is SU2, an open-source collection of C++ based software tools for performing Partial Differential Equation (PDE) solution and solving PDE-constrained optimization problems [6].
The toolset of SU2 is designed with Computational Fluid Dynamics (CFD) and aerodynamic shape optimization in mind but can be expanded to treat arbitrary sets of governing equations such as potential flow, elasticity, electrodynamics, chemically-reacting flows, and many others [6]. In particular, we will use the SU2_CFD tool, which is a code specifically designed for solving direct, adjoint, and linearized problems for the Euler, Navier-Stokes, and Reynolds-Averaged Navier-Stokes (RANS)

equation sets. It uses a Finite Volume Method (FVM); explicit and implicit time integration methods are available with Centered or Upwinding spatial integration schemes, it also has several advanced features to improve robustness and convergence, including residual smoothing, preconditioners, and agglomeration multigrid. For running a simulation with this tool the following 2 files are needed:

1) A configuration file (`.cfg`) which contains the parameters of the simulation.

2) A mesh file that contains the computational grid on which the simulation will be performed. Unfortunately, SU2 has not a dedicated mesher, so another software to produce the mesh is needed. Hence, Gmsh has been used, an open-source software that will be described in the next paragraph.

The simulation is run on the command line interface, and it produces different outputs as results:

1) A `.csv` file containing the convergence history of different parameters specified in the configuration file.

2) A `.vtu` file containing all the images and graphic visualization of the computational domain.

3) A second `.vtu` file containing only the graphic visualization of the flow around a specific surface (that has to be specified in the configuration file).

By default, SU2 is installed for serial computing, which means that only 1 core of the CPU is used to perform all the calculations, for very extensive and complex simulations this could lead to extremely long computational times. To prevent this issue there's the possibility to install additional tools for parallel computing which permit the distribution of the computational effort on different cores, reducing in this way the time needed to perform the simulation.

Gmsh

Gmsh is an open-source 3D finite element mesh generator with a built-in CAD engine and post-processor, its design goal is to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities [1].
Gmsh is built around four modules: geometry, mesh, solver and post-processing, the specification of any input to these modules is done either interactively using the graphical user interface, in ASCII text files using Gmsh's native scripting language (`.geo` file), or using the C++, C, Python or Julia Application Programming Interface (API) [1].
The main reason behind this choice is that Gmsh gives the possibility to export the `.geo` file, containing the mesh written the quite simple Gmsh native scripting language, to different formats, including the `.su2` extension that is used by SU2.

Analysis of the Fluid Dynamic Problem

The first step to correctly define a CFD simulation is the analysis of the fluid dynamic problem that has to be numerically solved.
In order to have comparative results, a common test case used for validation of transonic CFD codes has been chosen: the "Airfoil RAE 2822 - Pressure Distribution, and Boundary Layer and Wake Measurements" from the AGARD Advisory Report 138 [67].
The AGARD Advisory Report 138 is a collection of different fluid dynamic experimental tests written to aid in the development and refinement of computational methods and to improve their applicability and compatibility. The report presents selected test results and detailed geometric descriptions of representative airfoil, wing, wing-body and body-alone configurations [67].
Section 6 of this report contains examples of experiments made in a wind tunnel on a RAE 2822 airfoil, presented to give a range of conditions from wholly subcritical flow to conditions where a comparatively strong shock wave exists in the flow above the upper surface of the aerofoil (supercritical flow) [25].
For the preliminary discussion of the fluid dynamic problem let's consider the free-stream conditions (which are the flow conditions far from the airfoil) described in Test Case 6:

$M_\infty = 0.725$

$Re_\infty = 6.5 \cdot 10^6$

$\alpha = 2.92$ [°] (which is the angle of attack of the flow with respect to the airfoil)

The conditions above define a transonic flow field (transonic regime starts around $M_\infty = 0.7 \div 0.8$, the exact value depends on the object's critical Mach number), which is also certainly turbulent (very high Reynolds number). With this free-stream condition and with an incidence of 2.92° we should also expect the presence of a shock wave on the upper part of the airfoil.
This happens because the air flowing around the upper side of the airfoil, where the slope is positive, increases its velocity (because pressure decreases) until the max camber point, where the slope changes in sign, after this point the pressure begins to increase and the air velocity decreases, so a compression happens. If the free-stream velocity is high enough (generally around $M_\infty = 0.7$), the flow on the upper part of the airfoil becomes locally supersonic (the so-called "supersonic pocket" where locally $M \geq 1$, Fig. 4.1) and when the compression occurs a shock wave is generated.
The airfoil used in the simulation is the RAE 2822 super-critical airfoil. Super-critical airfoils are specifically designed to better perform in the transonic regime.
The reason why the supercritical airfoils are optimized for this specific flight regime

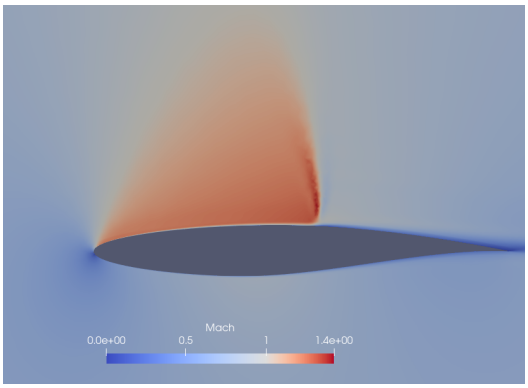

Figure 4.1. Visualization of the "supersonic pocket" on the airfoil at $M_\infty = 0.729$

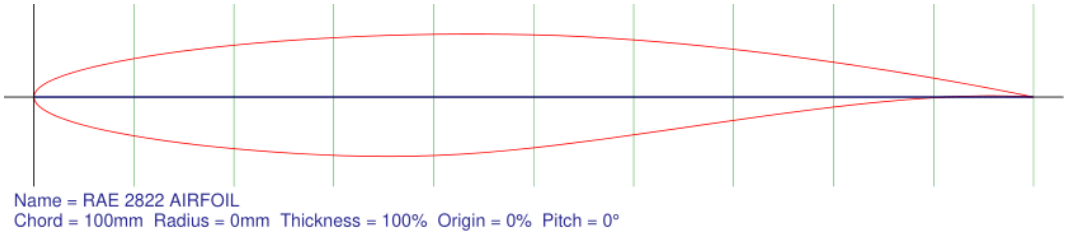

Figure 4.2. RAE 2822 Transonic Airfoil [65]

is mainly their high critical Mach number (M_{cr}), that is the free-stream Mach number (M_∞) for which a shock wave appears on the upper side of the airfoil, where the camber is at its max. In general shock waves on the airfoil lead to higher drag due to the high dissipative nature of this phenomenon, so for aircraft that are designed to have a cruise speed in this regime, these types of airfoils that mitigate this effect are fundamental. Another issue related to this regime is that the shock wave could lead to an anticipated separation of the boundary layer downstream, causing asymmetries and unsteadiness of the flow around the airfoil.
As previously mentioned, the mitigation is reached by increasing the critical Mach number and this is achieved by designing the airfoil with a low camber in the upper side that causes a lower increase in velocity of the airflow leading to a delay in the formation of the shock wave. The downside of this particular design is that the lower camber on the upper part of the airfoil causes an increase in the angular momentum of the airfoil, this means that in an aircraft designed with this type of airfoil there's the need to compensate for this stronger momentum with the horizontal stabilizer, leading in an overall decrease of the lift generated.
As already mentioned, this analysis is referred to the test case 6 of the AGARD report, but for this project it's necessary to define several simulations all distinguished by the aerodynamic incidence (the angle α) and the free stream conditions (Mach

number, M_∞, and Reynolds number, Re_∞). Therefore it's necessary to define the range of variation for these 3 parameters in order to define an appropriate number of simulations but all with the same kind of fluid dynamic problem to maintain the same 'core' for all the simulations.
The range of values chosen for α, M_∞ and Re_∞ for defining the different simulations are:

1) $\alpha = 0 \div 3.0$ [deg] with steps of $\Delta\alpha = 0.2$ [deg], the value of the aerodynamic incidence has been chosen always positive (to prevent possible issues with the flow field in the lower part of the airfoil for negative α) and with a max value of $\alpha = 3$ [deg] in order to always stay in the linear range of the curve $C_L = C_L(\alpha)$ of the RAE2822 Airfoil, see Fig. 4.3:

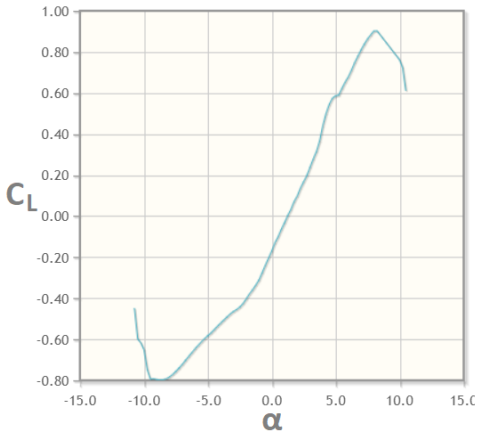

Figure 4.3. $C_L = C_L(\alpha)$ curve at $M_\infty = 0.0$ and $Re_\infty = 10^5$ for the RAE2822 Airfoil [66]

From Fig. 4.3 it's possible to notice that the linear relation between C_L and α extends from $\alpha \simeq -1.5 \div 4.5$ [deg] but, since the $C_L = C_L(\alpha)$ curve is obtained at fixed values of M_∞ and Re_∞, in order to have room for possible variation of the curve at different M_∞ and Re_∞, the value on $\alpha = 3$ [deg] has been used.

2) $M_\infty = 0.60 \div 0.80$ with steps of $\Delta M_\infty = 0.01$, these values of Mach number ensures the flow to be in the compressible regime (that starts around $M_\infty \simeq 0.3$) and are sufficiently high to develop a transonic flow around the airfoil.

3) $Re_\infty = 10^5 \div 10^7$ with steps of $\Delta Re_\infty = 4.95 \cdot 10^5$ this range for the Reynolds number should ensure that the flow around the airfoil is always fully turbulent, in order to use the RANS set of governing equations for all the simulations.

Summarizing, we have 16 values for α, 21 values for M_∞ and 21 values for Re_∞; in order to fully capture the influence of these three input parameters on the CFD simulations results it's necessary use all the possible combinations, therefore the final database will contain $16 \times 21 \times 21 = 7056$.

The mesh

Once the fluid dynamic problem has been discussed and analyzed the next step is to define a proper mesh for the simulations, this step is essential because the quality of the mesh, as already mentioned in section 2.3, dictates how fast the solution will converge and the quality of the results obtained. Since it's impossible to define a tailored mesh for each one of the 7056 CFD simulations that will be run it's necessary to design a generic mesh suitable for all the simulations, the key aspects of the thought process behind the definition of this mesh were:

1) **Simplicity**: Due to the very large number of simulations, checking all the results of the simulations one by one will be impossible, for this reason, it's necessary to have a simple mesh that ensures robustness to the solutions in order to achieve almost certain convergence for all the simulations run.

2) **Convergence velocity**: Since the software SU2 was installed for serial computing, in order to obtain the results of the simulations within an acceptable computational time, the mesh was designed to be not extremely refined where is not strictly necessary.

In the next paragraphs will be discussed the different aspects and features of the 2-dimensional mesh used for the simulations.

Geometrical Domain

The first aspect for the creation of a mesh is the Geometrical domain, which consists in the definition of the external boundary and all the geometrical entities within it. In this case the geometrical domain consists of a C-shaped external domain with the RAE2822 Airfoil in its center, Fig. 4.4 and 4.5.
The external domain has dimensions $x \in [-150,\ 150]$ and $y \in [-150,\ 150]$ while the airfoil has dimensions $x \in [0,\ 1]$ and $y \in [-0.1,\ 0.1]$ (all dimensions given in meters [m]). We can notice that the external domain is way larger than the airfoil, this is an important aspect because, in the subsonic regime, the flow conditions in a certain point of the domain also influence the flow conditions upstream, so the presence of the airfoil in the center of the domain will influence the flow upstream. Therefore it's very important to have a sufficiently large external domain where is plausible to consider the flow in the free-stream condition in order to correctly apply the free-stream boundary conditions.

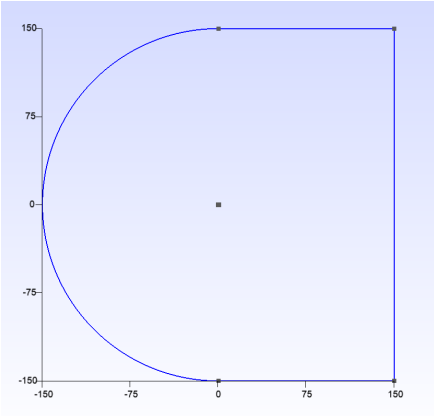

Figure 4.4. External Boundaries

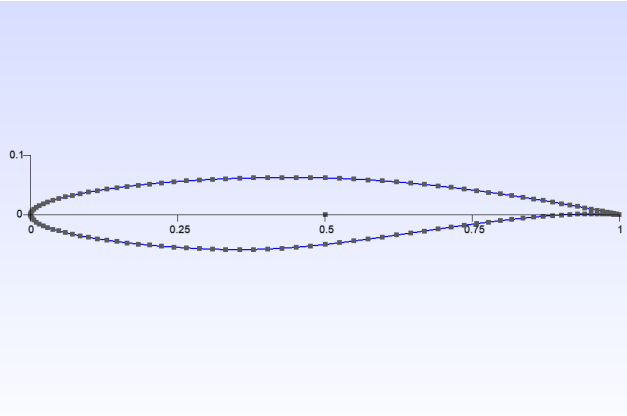

Figure 4.5. RAE2822 Airfoil

Physical Domain

The physical domain consists of the definition of the various physical entities on which is possible to apply the boundary conditions. The physical entities for this mesh are: the external boundary of the geometrical domain, also called "Farfield", where the free-stream boundary conditions are applied (M_∞, Re_∞ and T_∞) and the airfoil where the adiabatic wall boundary condition is applied (adherence condition at the wall, $\mathbf{v}_w = 0$, and no heat exchanges between the wall and the fluid). The actual physical domain, where the mesh will be generated, consists in the space between the Farfield and the airfoil.

Boundary Layer

Since the simulations involve viscous flow there's the necessity to carefully discretize the region of the boundary layer (close to the airfoil's wall) which is the region where the viscous effects are dominant. As previously mentioned in section 2.3, the most effective way to discretize the boundary layer is through a structured quadrilateral mesh with very thin elements near the wall, the elements have to be as much perpendicular as possible to the walls in order to fully capture the high gradients that characterize this region, see Fig. 4.6.
The boundary layer in the mesh has a constant thickness of 0.01 [m], the first element in the outer portion of the boundary layer has a y-dimension of $2.5 \cdot 10^{-3}$ [m], the size of the elements decrease exponentially with a factor of 1.2, reaching a y-dimension for the first cell near the wall of 10^{-5} [m]. With these dimensions the cells near the wall have an average value of normalized wall distance of $y^+ = 2.03$, for turbulence models with no law of the wall (like the Spalart-Allmaras model) the ideal value would be $y^+_{\text{id}} = 1$ but the previous value should give a good level of accuracy for the flow in the boundary layer despite not being ideal.

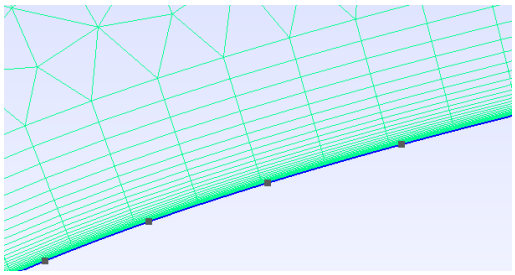

Figure 4.6. Boundary layer structured mesh near the RAE2822 airfoil's wall

Outer Grid

Outside the boundary layer, where the viscous effects become negligible, the mesh is unstructured and made of triangular elements created with a Delaunay triangulation[1] algorithm, which is the simplest meshing algorithm for triangular elements. The unstructured grid increases the numerical robustness by adding artificial diffusion (Section 2.3), the triangular elements, despite not being optimal for CFD simulations, are the most simple to define and solve.

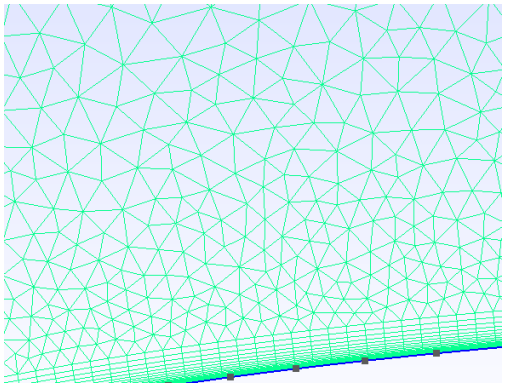

Figure 4.7. Mesh outside the boundary layer

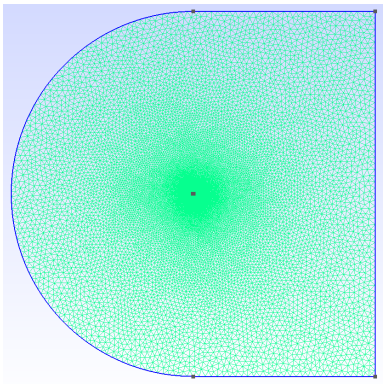

Figure 4.8. Overall view of the mesh

The elements have a characteristic dimension of 10 [m] near the external boundary of the computational domain (in this region the flow is uniform, therefore there's no need for particular accuracy), the size of the cells decrease linearly up to a characteristic dimension of $5 \cdot 10^{-3}$ [m] near the airfoil.

[1]Named after Boris Delaunay in 1934, a Delaunay triangulation, for a given set P of discrete points, is a triangulation DT(P) such that no point in P is inside the circumscribed circle of any triangle in DT(P) [27]

Statistics

The elements of the grid have an average aspect ratio $\gamma = 0.9421$ (the optimal value is $\gamma_{\text{id}} = 1$); the exact distribution of the aspect ratio for all the elements of the grid is reported in Fig. 4.9.

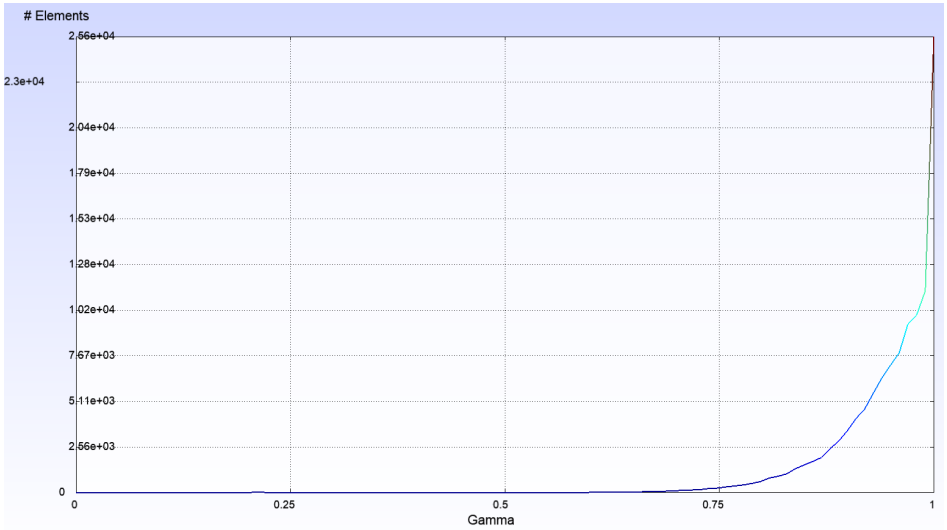

Figure 4.9. γ distribution between the elements of the grid

The average SICN (Signed Inverse Condition Number) is SICN $= 0.8589$ (the optimal value is $\text{SICN}_{\text{id}} = 1$), with the global distribution reported in Fig. 4.10:

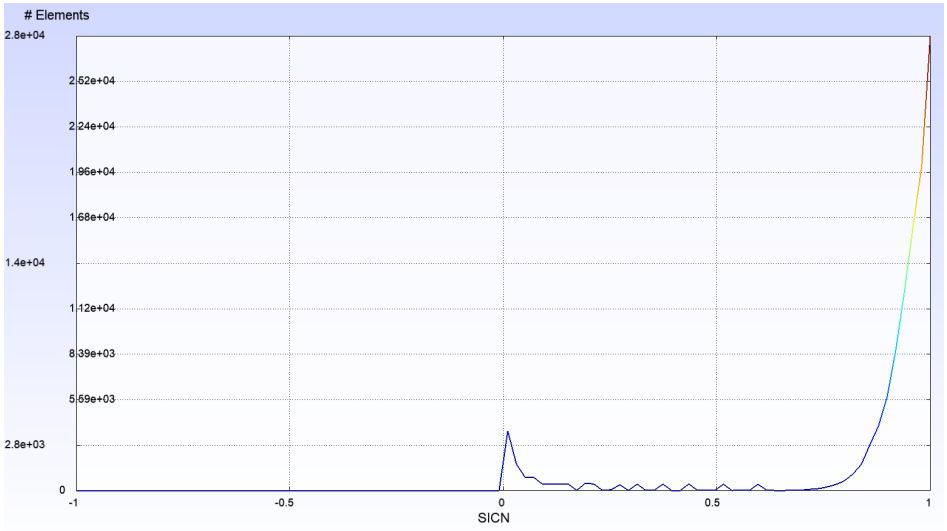

Figure 4.10. SICN distribution between the elements of the grid

In conclusion, the mesh presents good values of the quality indicators (γ, SICN and y^+), despite the approach was aimed towards robustness and simplicity, and it's formed by 101182 triangular elements and 12589 quadrilateral elements (in the boundary layer), for a total of 113771 elements.

Simulation Setup

Once the mesh is completed, the next step is to set the simulation parameters in the CFD software: Fluid properties, Governing equations, Turbulence model, Convergence criteria, etc. As already mentioned these settings and parameters are contained in the configuration file (`.cfg` file) that has to be given to the tool SU2_CFD as input.

Governing Equations

From the preliminary analysis made in Section 4.1, it's clear that the problem involves a fully turbulent flow, therefore we choose the RANS set of equations (Section 2.1).
The closure problem of the RANS set of equations is solved using a turbulence model and, for the simulations, the choice is the one-equation model of Spalart-Allmaras (Section 2.2).

Fluid Properties

The first properties that have to be set are the free-stream conditions, these parameters define the condition of the fluid in the Farfield, where the flow field is uniform and steady.
As already anticipated the various simulations of the database differ from each other for the free-stream conditions, dictated by the three parameters α, M_∞ and Re_∞. These three parameters are not enough to fully compute all the fluid properties in the free-stream, we need to assume the value of an additional thermodynamic variable to fully determine the free-stream condition, in this case, the choice is the free-stream static temperature for which the value of $T_\infty = 460\ [R] \equiv 255.556\ [K]$ has been assumed, this value will remain constant between all the simulations. Once the values of M_∞, Re_∞ and T_∞ are known the software can define all the other thermodynamic variables like static pressure, density, total pressure, etc. using the definition of Mach number, Reynolds number and the gas law.
Since the fluid problem involves viscosity, it's also necessary to define a viscosity model, it's possible to use a constant viscosity model that keeps the same value for all the flow field but it would be an inaccurate method since in reality viscosity depends on the local temperature of the fluid and, for an airfoil simulation in the transonic regime, near the airfoil the temperatures changes rapidly, especially near the stagnation point on the leading edge. For this reason the viscosity model

adopted is the Sutherland's law viscosity model [70] which relates the local value for dynamic viscosity to the value of the local static temperature of the fluid:

$$\mu = \mu_{\text{ref}} \left(\frac{T}{T_{\text{ref}}} \right)^{\frac{3}{2}} \frac{T_{\text{ref}} + S}{T + S} \tag{4.1}$$

where T_{ref} is an arbitrary reference temperature and μ_{ref} is the corresponding dynamic viscosity, S is the Sutherland's coefficient and T is the local static temperature. For the simulation were chosen the following values for the references and the coefficient:

Quantity	**Value**
T_{ref}	$288.15\ [K]$
μ_{ref}	$1.789 \cdot 10^{-5}\ [Pa\ s]$
S	$110.4\ [K]$

Table 4.1. Sutherland's law coefficient and reference values

Boundary Conditions

It's necessary to specify to the software which physical entity of the mesh has a boundary condition and which kind of boundary condition is.
The boundary condition of the simulations are:

- **Farfield condition** in all the **external domain**: in the outer boundary of the physical domain the flow is considered in its free-stream condition.
- **Adiabatic wall condition** for the **airfoil**: for all the surfaces of the airfoil the no-slip condition ($\mathbf{v}_w = 0$) is imposed, together with and the condition of no heat exchanges between fluid and wall.

Parameters for the numerical approximation

In this paragraph the choices behind the definition of the numerical solution method for the simulations are described and discussed.

- **Convective discretization scheme**: For the approximation of the convective term of the governing equation the Jameson-Schmidt-Turkel (JST) central scheme has been chosen, already mentioned in Section 2.3. The values for the 2-nd and 4-th order artificial dissipation coefficients are $\epsilon^{(2)} = 0.00$ and $\epsilon^{(4)} = 0.02$ (which are the default values assumed by the software).
- **Turbulent convective discretization scheme**: The software only allows the option for a 1-st order Scalar-Upwind scheme for the turbulent convective term.

- **Time-stepping scheme**: The time-advancing scheme for the iteration adopted (this is a steady simulation, so the solution is not time-dependant, the time scheme is used to define how to advance in the iteration) is the Implicit Euler scheme because it allows a higher CFL number, which has been chosen equal to CFL = 45, after some investigations aimed to determine the max CFL number that would give a stable solution.

- **Linear solver**: For the solution of the linear system obtained after the discretization of the governing equations the Flexible Generalized Minimal Residual (FGMRES) linear solver is used (which is the default option for the software), it is a common iterative method for solving indefinite non-symmetric system of linear equations [47].

- **Stopping criteria**: To stop the simulations 2 possible stopping-criteria were chosen, as soon as one of them is satisfied the simulation will end.

    1) **Convergence criteria**: Since the simulations are focused on computing the aerodynamic coefficient C_L and C_D the convergence criteria adopted is the Cauchy residual approach applied to the two aerodynamic coefficients. This approach determines the relative difference between the values of the coefficients between two consecutive iterations (this is called Cauchy element), the convergence criteria is satisfied when the average over a certain number of Cauchy elements (for which the default value of 100 elements has been set) is smaller than the criteria imposed (which has been set equal to $\epsilon = 10^{-4}$)
    2) **Max iteration criteria**: To prevent the simulations to run for an infinite time if the solution doesn't converge, the simulation is forced to stop if the solver reaches 3000 iterations.

4.2 Prediction of the Aerodynamic Coefficients

Once the results of the 7056 CFD simulations are obtained the next step is to implement a Multi-Layer Perceptron that will predict the two Aerodynamic coefficients computed for every set of input parameters α, M_∞ and Re_∞.
All the neural networks made during this thesis project were developed using Python 3.9.2 with prevalent usage of the libraries NumPy [31], which is a library that implements numerical tools and functions for scientific computing, particularly suitable to work with arrays, and PyTorch [48], which is a library that introduces tools to develop and work with Machine Learning.
The focus of this first MLP is to learn the relations between the free-stream conditions of the flow and the aerodynamic coefficients C_L and C_D by interpolation.
The database for this Neural Network has the structure reported in Table 4.2.

INPUT FEATURES	TARGETS
$[\alpha,\ M_\infty,\ Re_\infty]_1$	$[C_L,\ C_D]_1$
$[\alpha,\ M_\infty,\ Re_\infty]_2$	$[C_L,\ C_D]_2$
$\vdots$	$\vdots$
$[\alpha,\ M_\infty,\ Re_\infty]_N$	$[C_L,\ C_D]_N$

Table 4.2. Database for the aerodynamic coefficients, $N = 7056$ samples

The first step for the implementation of the neural network is splitting the overall database into 3 sets of samples: Training set, Validation set and Test set.

Preparation of the Database

The 3 subsets of the database achieve different tasks:

- **Training set**: This set contains all the samples that will be used for the actual training of the NN.
- **Validation set**: This set is used to obtain an evaluation of the model during the training, in order to visualize, for example, possible increase of the loss due to overfitting[2] which can't be seen in the training process.
- **Test set**: This set is used after the training to visualize the results of the final model.

A common subdivision for the 3 subsets, as percentage of the overall dataset, is: **Training** 70%, **Validation** 20% and **Test** 10%.

Overall	Training	Validation	Test
7056	4939	1270	847

Table 4.3. Number of samples for the different subsets (prediction of the aerodynamic coefficients)

Another necessary process that has to be made on the database is Standardization, looking at the magnitude of the input features we find out that $\alpha \simeq 10^{-1} \div 10^{0}$ and

[2]Overfitting happens when the training set has too many samples, or the network architecture is too complex, for the actual complexity of the function that has to be learned.
This problem can lead to the interpretation of the noise in the training samples by the network's model as actual training data, affecting the performance of the NN and its ability to correctly generalize [12]

$M_\infty \simeq 10^{-1}$ have a similar order magnitude, but $Re_\infty \simeq 10^5 \div 10^7$ has several orders of magnitude more than the other two parameters. This will cause the network to overestimate the importance of Re_∞ as input feature and basically neglecting the other two, leading to bad results in the training. To prevent this problem it's necessary to standardize the training, validation and test input features in order to have the same order of magnitude for all the input features.
Having a n number of samples, the normalized value for the i-th sample is:

$$x^i_{\text{stand}} = \frac{x^i - x_{\text{mean}}}{x_{\text{std}}} \tag{4.2}$$

where x^i is the non-standardized sample, x_{mean} is the mean value of the n samples and x_{std} is the standard deviation of the n samples, which is given by:

$$x_{\text{std}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x^i - x_{\text{mean}}\right)^2}$$

The standardization of the 3 subsets is made with respect to the mean value and the standard deviation of the **Training set** only. This because it's the training set that contains the samples on which the networks learns the pattern that relates the input to the output.

Cost Function

As already mentioned in Section 3.2, the function chosen for the evaluation of the Cost (or Loss) during the training process depends on the specific task of the NN, since this is a regression task the cost function adopted is the Mean Squared Error (MSE), see equation (3.16), which is a common choice for this kind of networks:

$$C_{\text{MSE}} = \frac{1}{N}\sum_{i}^{N}\left(\mathbf{a}^L - \mathbf{y}(\mathbf{x})\right)^2$$

Training Algorithm

The training algorithm adopted in this first MLP is the Adam Algorithm (see Section 3.2) with an adaptive learning rate η that starts from a value of $\eta_{\text{start}} = 10^{-2}$ and if the Loss doesn't decrease for a total of 10 iterations (this parameter is called Patience [20]) the learning rate is reduced by a factor $k = 0.1$.

Hyperparameters

The hyperparameters are the network's not-trainable features that can be tuned to increase the ANN's performances. They can be different from task to task, for this Multi-Layer Perceptron the set of hyperparameters chosen is:

- **Network Architecture**: Number of layers and number of nodes (neurons) for each layer, type of activation function.
- **Batch size**: Number of samples of the training set taken for each weights and biases update iteration, a larger batch size leads to faster but less accurate training, a smaller one leads to more accuracy but the cost function decreases with a more oscillatory behaviour and it's computationally more expensive.
- **Number of Epochs**: Number of training iterations, all the training made for testing the hyperparameters' influence have been run on a fixed number of 500 epochs.

Before trying to predict the two aerodynamic coefficients simultaneously, in order to use a "step-by-step" approach, we tried to predict the two coefficients separately on two different MLPs with the same features and algorithm described before and then use a third MLP to predict the two coefficient together.
To explore the influence of the hyperparameters on the performance of the 3 neural networks, a grid search made on different values of the batch size and different architectures has been performed. In order to have a statistical basis that ensures the reliability of the results obtained, 10 training sessions have been performed for every combination of batch size and network architecture.

Batch size

To test the influence of the batch size 4 values were chosen:

Batch Size	25	50	100	150

Table 4.4. Batch size values used for the grid search

The expectation is to achieve the highest accuracy with a batch size of 25 but with a longer training time, on the other hand with a batch size of 150 we expect the lowest accuracy and the lowest training time. The focus is to find a trade-off between accuracy a computational cost in order to determine the optimal value for the batch size.

Network Architecture

All the 3 MLPs have the same input layer with 3 neurons (α, M_∞ and Re_∞) and the same structure for the hidden layers, the only difference between the networks that predict the value of the single aerodynamic coefficient and the network that predicts the values of both C_L and C_D is the dimension of the output layer, for the single-coefficient MLPs the output layer has a dimension of 1 (C_L or C_D), for the double-coefficient MLP the dimension of the output layer is 2 (both C_L and C_D). All the hidden layers have ReLU activated neurons, the output layer has only linear activated neurons, which means that the activation function for the last layer is linear and doesn't change the output of the neuron. We tested different possible architectures with a different number of hidden layers and size, in order to find the best model to approximate the aerodynamic coefficients.

- **Single Hidden Layer Architecture**: This is the simplest architecture tested, it consists of the input layer, a single hidden layer and the output layer.

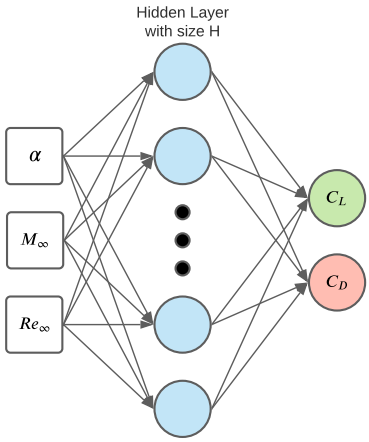

Figure 4.11. Single Hidden Layer Architecture

  For this type of architecture two possible dimensions H for the hidden layer have been tested, 40 neurons and 100 neurons.

- **Funnel Architecture**: With this kind of architecture the structure resembles the shape of a funnel, the first hidden layer has the highest number of neurons and the layers shrink up to the dimension of the output, this is a common architecture for regression MLP.
  For this architecture different sizes of the layers and different numbers of hidden layers have been tested: 2 hidden layers with 40 and 20 neurons, 2 hidden

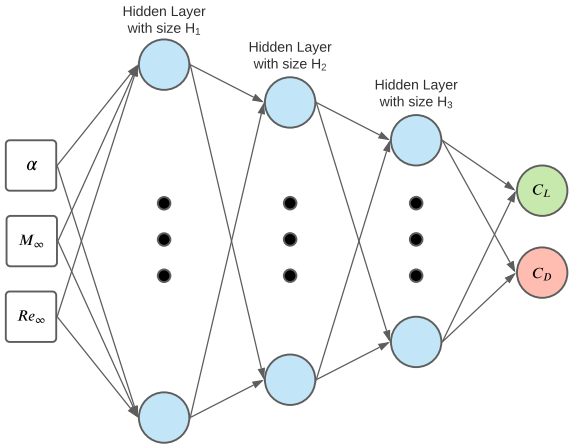

Figure 4.12. Funnel Architecture, where the hidden layers dimension are $H_1 > H_2 > H_3$

layers with 100 and 10 neurons, 3 hidden layers with 80, 40 and 20 neurons and the last one with 4 hidden layers with 160, 80, 40 and 20 neurons.

- **Rhombus Architecture**: The concept behind this network architecture is to gradually increase the complexity of the model before shrinking to the output dimension as in the Funnel Architecture.

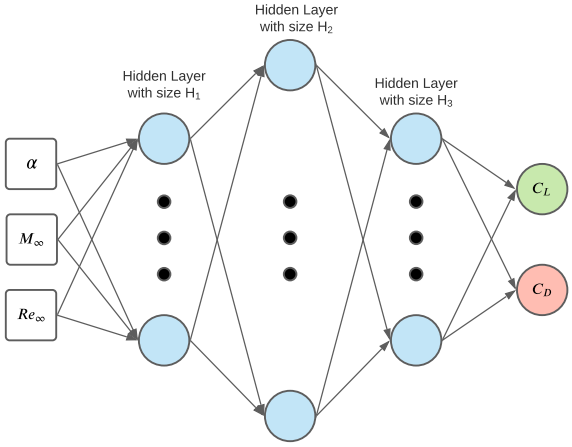

Figure 4.13. Rhombus Architecture, where the hidden layers dimension are $H_2 > H_1$ and $H_2 > H_3$

The architectures tested in the grid search are: 3 hidden layers with 20, 40 and 20 neurons, 5 hidden layers with 20, 40, 80, 40 and 20 neurons and the

most complex model tested in the grid search, with 7 hidden layers with 20, 40, 80, 160, 80, 40 and 20 neurons.

As a summary, network architectures explored for this first type of Multi-Layer Perceptron are reported in Table 4.5.

Hidden Layers						
H_1	H_2	H_3	H_4	H_5	H_6	H_7
40						
100						
40	20					
100	10					
80	40	20				
160	80	40	20			
20	40	20				
20	40	80	40	20		
20	40	80	160	80	40	20

Table 4.5. Architectures for the Aerodynamic Coefficients MLP

With 4 batch sizes and 9 architectures, there are 36 combinations of the 2 hyper-parameters, every combination is trained 10 times for statistical purposes, so for each one of the three grid searches performed (two for the single coefficient MLPs and one for the double coefficient MLP) 360 training processes have been performed.

4.3 Prediction of the Flow Field

For the MLP that predicts the flow field around the airfoil, the database consists of the same 3 input free-stream conditions (α, M_∞ and Re_∞) but the targets are the images of the flow field near the RAE2822 in terms of local Mach number of the flow.
The images are a product of the CFD simulations as `.vtu` files, these files contain all the flow variables for each cell-center in the computational domain. In order to reduce the size of the problem we decided to take a 'snapshot' (which produces an image of the solution) only of the region around the airfoil, where its influence on the velocity field of the flow is stronger. The reason behind this choice is that, as previously mentioned in Section 4.1, the mesh is constituted by a total of 113771 cells where the major part is in the region near the airfoil. Therefore if the actual solutions of the CFD simulations near the airfoil are used as output for the MLP the resulting number of neurons in the output layer could lead to an unsustainable

increase of computational cost. On the other hand, converting the actual solutions into images can give us control on the number of neurons in the output by increasing or decreasing the resolution of the images. This approach gives quite reliable results, despite loosing some accuracy of the CFD solutions in the conversion.

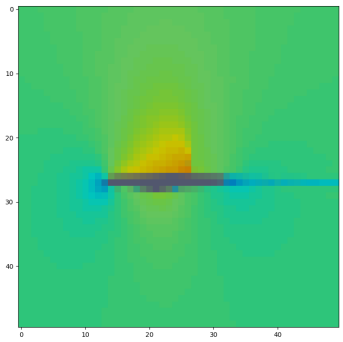

Figure 4.14. Target sample (`.png` image with resolution 50×50)

The images used are `.png` files with a resolution of 50×50 pixels (Fig. 4.14 (coloured), the reason for this low resolution is that, for predicting the images with a Multy-Layer Perceptron, every pixel becomes a variable of the output layer; this means that, considering a coloured image of 50×50 pixels, the dimension of the output layer is $50 \times 50 \times 3 = 7500$ neurons. Therefore, the network architecture necessary to this kind of output dimensions is thousands of times larger than the one needed for the aerodynamic coefficients where the output had only 2 neurons, so to avoid having computational power issues, we decided to use low resolution images as output.
The task for this neural network is to find the relation between the free-stream conditions and the value of the pixels of the relative image of the flow field. The database on which this MLP will be trained has the following structure: where the

INPUT FEATURES	**TARGETS**
$[\alpha,\ M_\infty,\ Re_\infty]_1$	$\left[\mathrm{px}_{(1,1,1)},\ \mathrm{px}_{(1,2,1)},\ \cdots,\ \mathrm{px}_{(50,50,3)}\right]_1$
$[\alpha,\ M_\infty,\ Re_\infty]_2$	$\left[\mathrm{px}_{(1,1,1)},\ \mathrm{px}_{(1,2,1)},\ \cdots,\ \mathrm{px}_{(50,50,3)}\right]_2$
$\vdots$	$\vdots$
$[\alpha,\ M_\infty,\ Re_\infty]_N$	$\left[\mathrm{px}_{(1,1,1)},\ \mathrm{px}_{(1,2,1)},\ \cdots,\ \mathrm{px}_{(50,50,3)}\right]_N$

Table 4.6. Database for the flow field (in terms of local Mach number), $N = 7056$ samples

first two indices of the pixels indicate the pixel position in x and y and the third

index indicates the color of the pixel in the RGB format (1 for the red, 2 for the green and 3 for the blue).
The overall database is split in training, validation and test sets in the same percentages of the aerodynamic coefficients one: **Training** 70%, **Validation** 20% and **Test** 10%. So the corresponding number of samples are the same of Table 4.2. The three subsets are then standardized with respect to the training set mean value and standard deviation in order to have the same order of magnitude for the three inputs.

Cost Function

The function chosen for the evaluation of the Cost (or Loss) during the training process is the Mean Squared Error (MSE), equation (3.16), since this is a regression-kind network and this is a suitable cost function for this task.

$$C_{\text{MSE}} = \frac{1}{N} \sum_{i}^{N} \left(\mathbf{a}^{L} - \mathbf{y}(\mathbf{x}) \right)^{2}$$

Training Algorithm

The training algorithm used for this neural network is the Stochastic Gradient Descent (SGD) algorithm (Section 3.2) with an adaptive learning rate η that starts from a value of $\eta_{\text{start}} = 10^{0}$ and gets updated, after 10 iteration with no-decreases in the loss, by a multiplying factor of $k = 0.1$.

Hyperparameters

As already done for the aerodynamic coefficients MLPs, we used a "step-by-step" approach for this second type of MLP: before trying to predict the coloured images we took a preliminary step and tried to predict the grey-scale version of the images, converting an RGB image in the grey scale means reducing the color layers from 3 to only 1, the results is an output layer with only $50 \times 50 = 2500$ neurons instead of the 7500 neurons of the coloured one, as represented in Fig. 4.15. The effect is a slightly reduced size of the hidden layers, leading to an overall decrease in the computational effort.
The only hyperparameter chosen for the test is the **Network Architecture**, this because the increased complexity of these networks led longer training times. A single training for this kind of networks can take from one to two hours (depending on the complexity of the model and the number of training epochs performed), but since in the grid-search several training sessions are performed the overall computational time needed grows rapidly, a single grid-search can takes days of computing on the HPC server using one Nvidia Tesla V100 GPU as computing device. Trying

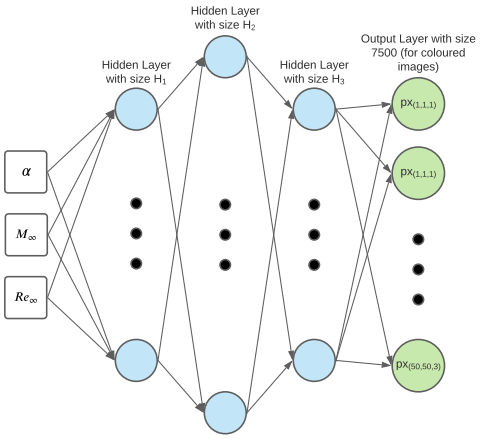

Figure 4.15. Representation of the network architecture of the MLP use for the reconstruction of the coloured images

to explore more hyperparameters with all the possible combinations would lead to unsustainable computational times on the server.
To test the influence of the network architecture on the two MLPs (grey-scale and coloured images) a grid search has been performed, all the training sessions were made with a batch size of 100 and for a total number of 2500 epochs. For every architecture 10 training sessions have been performed to have a statistical basis to evaluate the network performance.

Network Architecture

For both the grey-scale MLP and the color images MLP, all the hidden layers have ReLU activated neurons and the output layer has only linear activated neurons.
For the grey-scale MLP the architecture tested are:

- **Funnel Architecture**: Five different architecture for this kind of structure have been tested: 3 hidden layers with 5000, 5000 and 2500 neurons, 5 hidden layers with 5000, 5000, 5000, 2500 and 2500 neurons, 5 hidden layers with 10000, 10000, 5000, 5000 and 2500 neurons, 8 hidden layers with 10000, 10000, 10000, 5000, 5000, 5000, 2500 and 2500 neurons and the last one with 6 hidden layers with 15000, 12000, 10000, 7500, 5000 and 2500 neurons.

- **Rhombus Architecture**: For this kind of structure only two architectures have been tested: 5 hidden layers with 2500, 5000, 5000, 3000 and 2500 neurons and 6 hidden layers with 500, 1000, 2500, 5000, 3000 and 2500 neurons.

It's immediate to observe how the network architectures have more layers and more neurons with respect to the aerodynamic coefficients MLPs, this is a consequence

of the larger output dimensions.
The architectures tested in the grid search relative to the gray-scale MLP are summarized in Table 4.7

Hidden Layers							
H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
5000	5000	2500					
5000	5000	5000	2500	2500			
10000	10000	5000	5000	2500			
10000	10000	10000	5000	5000	5000	5000	2500
15000	12000	10000	7500	5000	2500		
2500	5000	5000	3000	2500			
500	1000	2500	5000	3000	2500		

Table 4.7. Architectures for the grey-scale MLP

For the color images MLP the architecture tested in the grid search are:

- **Funnel Architecture**: Three network architectures with this kind of structure have been explored: 3 hidden layer with 10000, 10000 and 7500 neurons, 5 hidden layers with 15000, 15000, 10000, 10000 and 7500 neurons and the last one with 8 hidden layers with 15000, 15000, 15000, 10000, 10000, 10000, 7500 and 7500 neurons.

- **Rhombus Architecture**: For this kind of structure only two architectures have been tested: 5 hidden layers with 7500, 10000, 10000, 8000 and 7500 neurons and 6 hidden layers with 1000, 2500, 7500, 10000, 8000 and 7500 neurons.

To summarize, the architectures tested in the grid search relative to the color images MLP are represented in Table 4.8.

Hidden Layers							
H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
10000	10000	7500					
15000	15000	10000	1000	7500			
15000	15000	15000	10000	10000	10000	7500	7500
7500	10000	10000	8000	7500			
1000	2500	7500	10000	8000	7500		

Table 4.8. Architectures for the color images MLP

Chapter 5

Results & Conclusions

In this conclusive Chapter, the results obtained during this thesis project are presented and discussed. The first Section contains the comparison between the results of the CFD simulation and the reference Test Case, the second Section consists in the display and analysis of the results of the grid search performed to explore the hyperparameters influence on the different NNs implemented in this project.

5.1 CFD Simulation: comparison with the reference Test Case

To analyze the quality of the CFD simulations, let's compare the simulation with the free-stream conditions of the reference Test Case 6, from AGARD Advisory Report AR138 [25], with the comparison results obtained from the NASA's NPARC (National Program for Application-oriented Research in CFD) validation program [57] on the RAE2822 airfoil, which is a collection of five studies on the RAE2822 airfoil, aimed to the validation of the WIND-US CFD code [72].
The free-stream conditions of the flow are:

- $\alpha = 2.31$ [deg]
- $M_\infty = 0.729$
- $Re_\infty = 6.5 \cdot 10^6$

with a free-stream static temperature of $T_\infty = 255.556$ $[K]$.

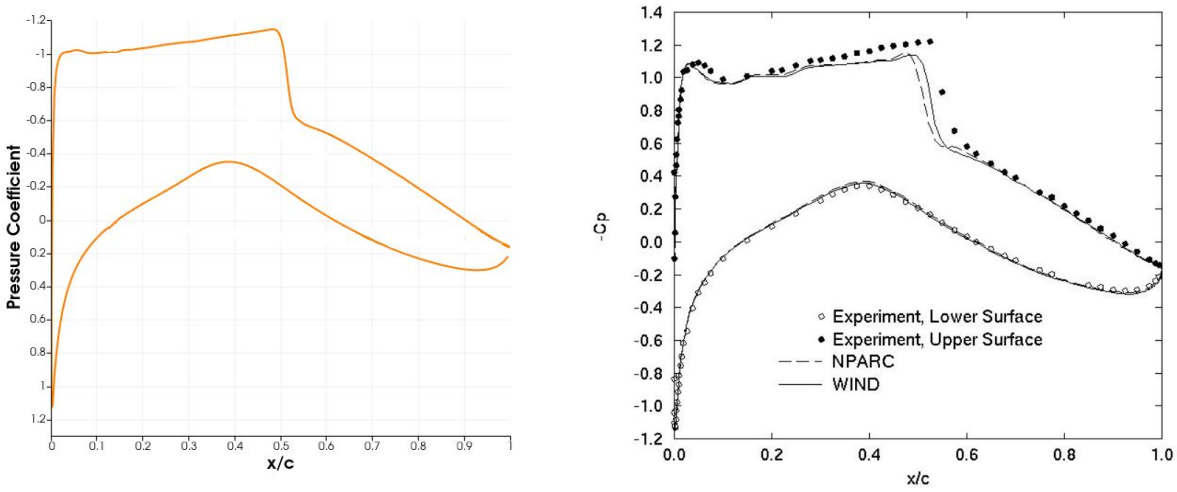

Figure 5.1. Pressure coefficient distribution around the airfoil, comparison between the results of the CFD simulation (left) and the reference [58] (right)

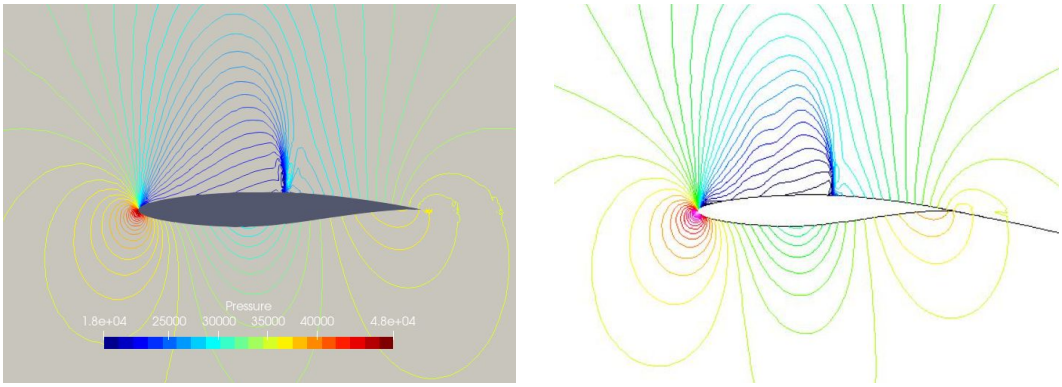

Figure 5.2. Pressure field around the airfoil, comparison between the results of the CFD simulation (left) and the reference [58] (right)

In Fig. 5.1 the pressure coefficient C_p distribution around the airfoil is represented, the upper curve is the C_p distribution on the upper part of the airfoil and is clearly visible, near $x/c \simeq 0.5$, the sudden increase of pressure due to the presence of the compression shock. Fig. 5.2 shows the distribution of the iso-pressure lines near the airfoil, in the zone where the iso-pressure lines stacks together we can see the value of the absolute pressure increasing suddenly for the effect of the shock. The comparison shows almost identical results, again ensuring the reliability of the results obtained from the CFD simulations performed.

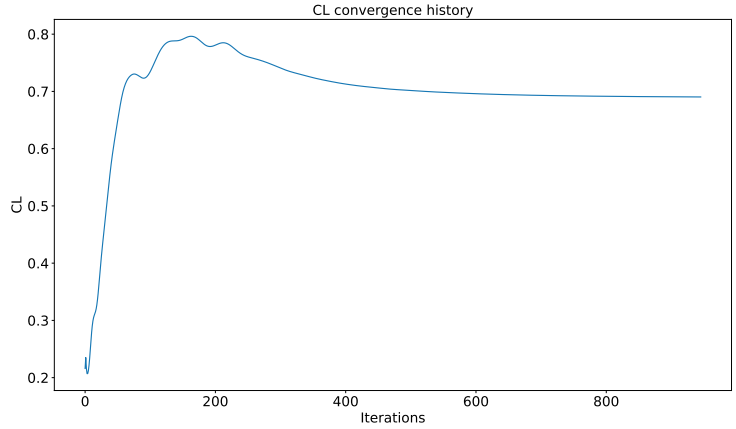

Figure 5.3. Convergence history of the lift coefficient C_L

Fig. 5.3 shows the convergence history of the lift coefficient C_L, the simulation has satisfied the convergence criteria in 945 iterations and the lift coefficient converged to a value of $C_L = 0.69$.

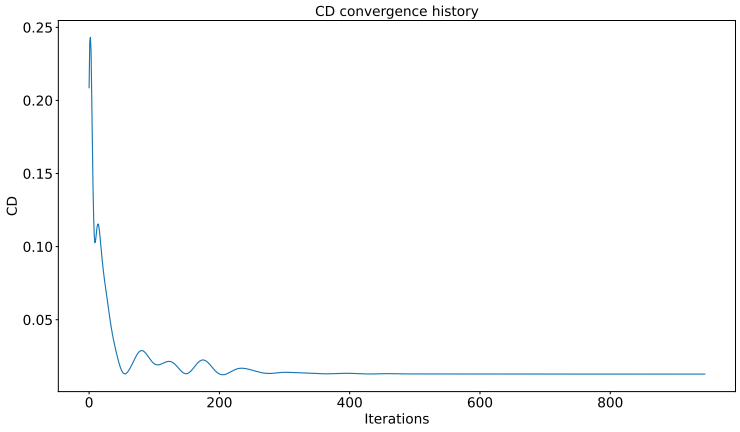

Figure 5.4. Convergence history of the Drag coefficient C_D

In Fig. 5.4 the convergence history of the drag coefficient C_D is represented, which has converged to the value $C_D = 0.0128$.

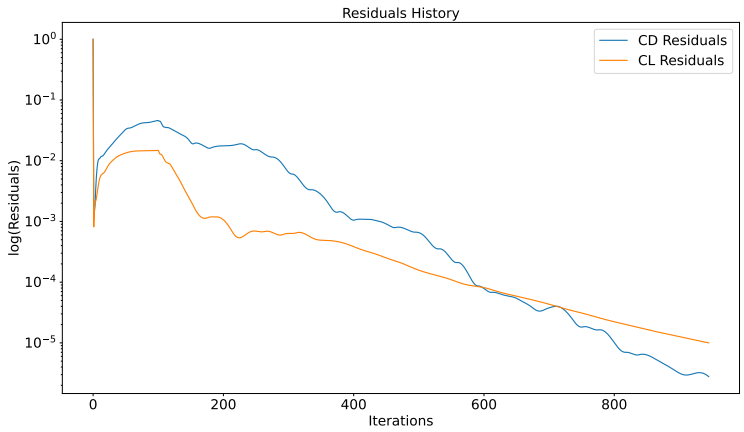

Figure 5.5. Residuals history of the two aerodynamic coefficients

Fig. 5.5 represents the residuals history for C_L and C_D, we can notice how the drag coefficient was the last to satisfy the convergence criteria for the Cauchy residual $\epsilon = 10^{-4}$ and it's convergence history is much more unstable compared to the C_L one, even after the initial transitory. This is due to the complexity of the several effects that influence the drag, such as the wave drag, the viscous drag related to the turbulent flow, etc.
The next figures represents the flow field around the airfoil, in term of local Mach number, for 6 CFD simulations with different free-stream conditions.

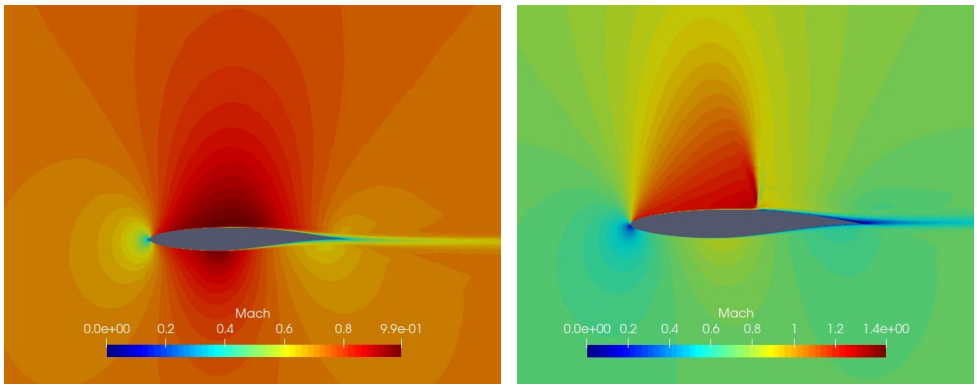

Figure 5.6. Low aerodynamic incidence $\alpha = 0.0$ [deg] (left) and High aerodynamic incidence $\alpha = 3.0$ [deg] (right) comparison

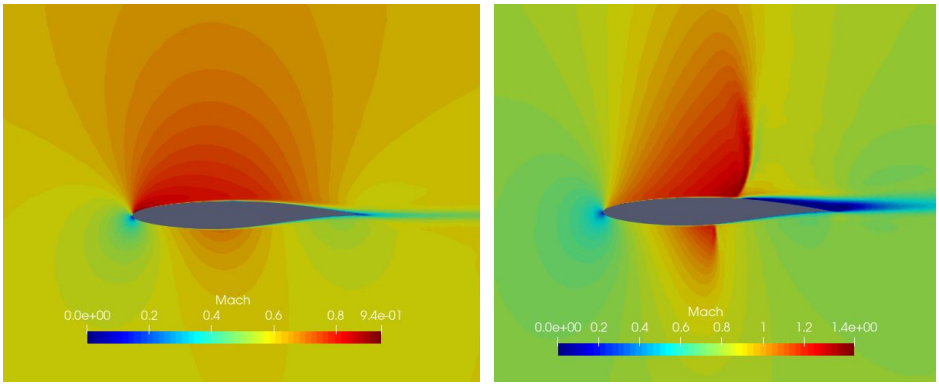

Figure 5.7. Low Mach number $M_\infty = 0.60$ (left) and High Mach number $M_\infty = 0.80$ (right) comparison

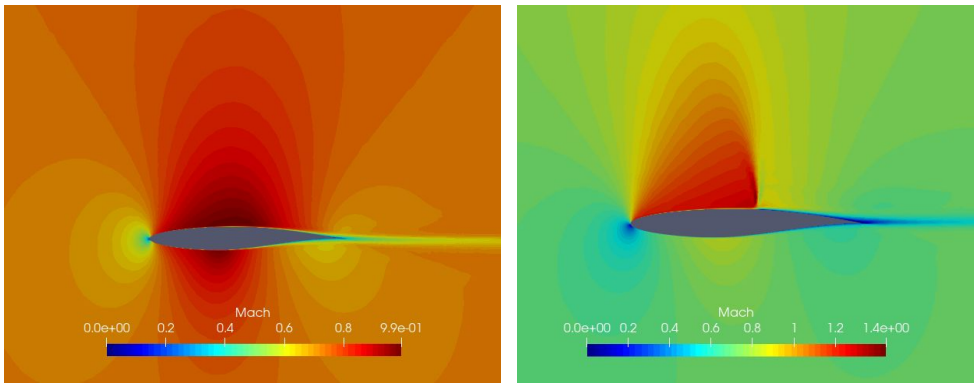

Figure 5.8. Low Reynolds number $Re_\infty = 10^5$ (left) and High Reynolds number $Re_\infty = 10^7$ (right) comparison

As we can see with low values of α, M_∞ and Re_∞ the flow doesn't reach the supersonic state on the upper-side of the airfoil, therefore the compression shock doesn't appear and the pressure steadily increases after the max camber point. On the other hand, for high values of α, M_∞ and Re_∞ we can see that the shock presents different configurations and, most important, it falls in different positions of the upper-part of the airfoil.

5.2 Results of the Aerodynamic Coefficients Multi-Layer-Perceptrons

The results of the various grid searches performed to explore the influence of the hyperparameters are summarized in box-plots that represent the statistical results, in terms of Mean Squared Error (MSE) Loss, for every combination of architecture and batch size explored.

Single Coefficient MLP - C_D

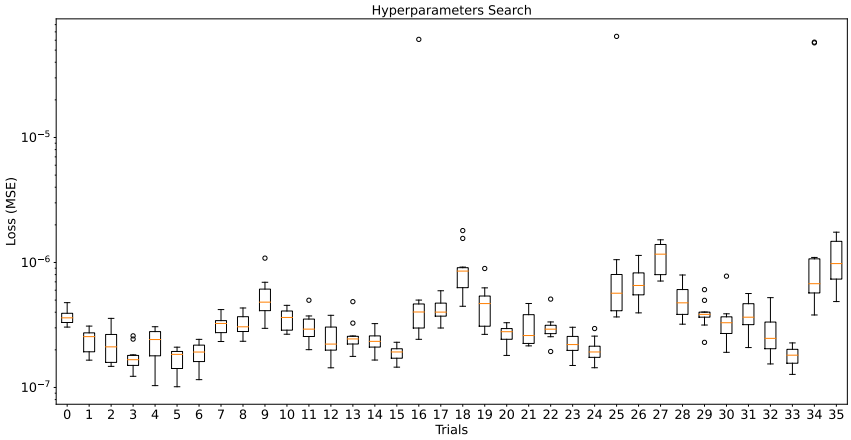

Figure 5.9. Grid Search results (not filtered), logarithmic scale - Single coefficient MLP (C_D)

Fig. 5.9 shows the box-plot that summarizes the results for the Single coefficient MLP for the drag coefficient, with the Loss axis in logarithmic scale. This plot is used as preliminary-check for results that present outliers with different order of magnitude for the Loss (like the trials 16, 25 and 34 in Fig 5.9) from the other ones, in order to delete these results in the final box-plot for clarity purpose.
After having filtered the unreliable results, Fig. 5.10 shows the final results for the first grid search performed on the MLP trained on the drag coefficient. The results are grouped by the batch size values and every box-plot color represents a different network architecture.
As we can see the results presents very low values of Loss (order of magnitude of $10^{-7} \div 10^{-6}$). As expected the best results in terms of Loss function are obtained with low batch size and, as we can see, the simple single hidden layer architecture (red and grey boxes) gave the worst results. In general, all the results obtained in this first grid search can be considered reliable and very accurate. The model with architecture 3, 80, 40, 20, 1 (green boxes in Fig. 5.10) and with a batch size of

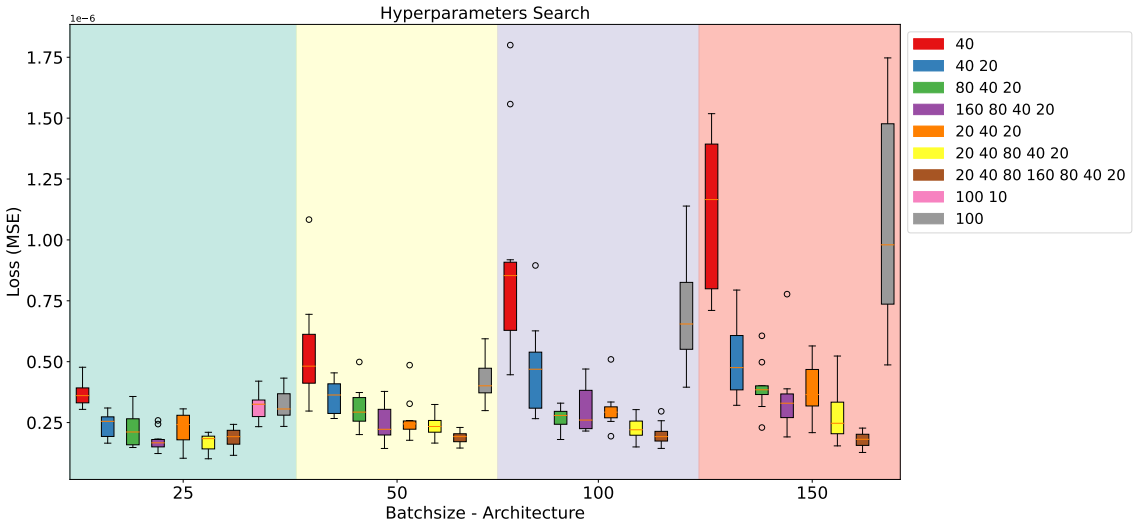

Figure 5.10. Grid Search results - Single coefficient MLP (C_D)

100 samples can be considered the 'best' trade off between computational cost and accuracy of the results. The following plots represent the detailed results of one of the 10 training sessions performed with this combination of hyperparameters.

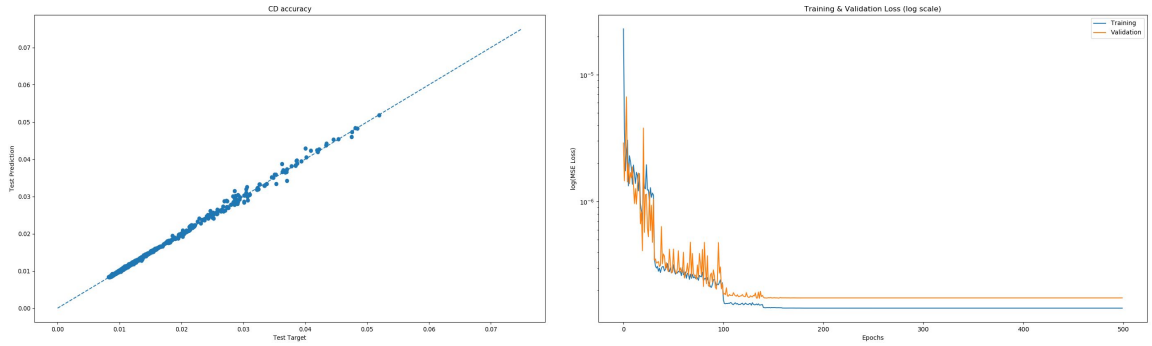

Figure 5.11. Accuracy of the Test prediction (left) and Loss vs Epochs (right)

The left plot in Fig. 5.11 represents the accuracy of the Test set predictions (predicted values vs target values), the closer the points are to the bisecting line, the more accurate the prediction is. The right plot represents the Loss decrease during the training (blue line) and the validation (orange line) for every epoch, the Loss axis is in logarithmic scale for clarity purpose.

Single Coefficient MLP - C_L

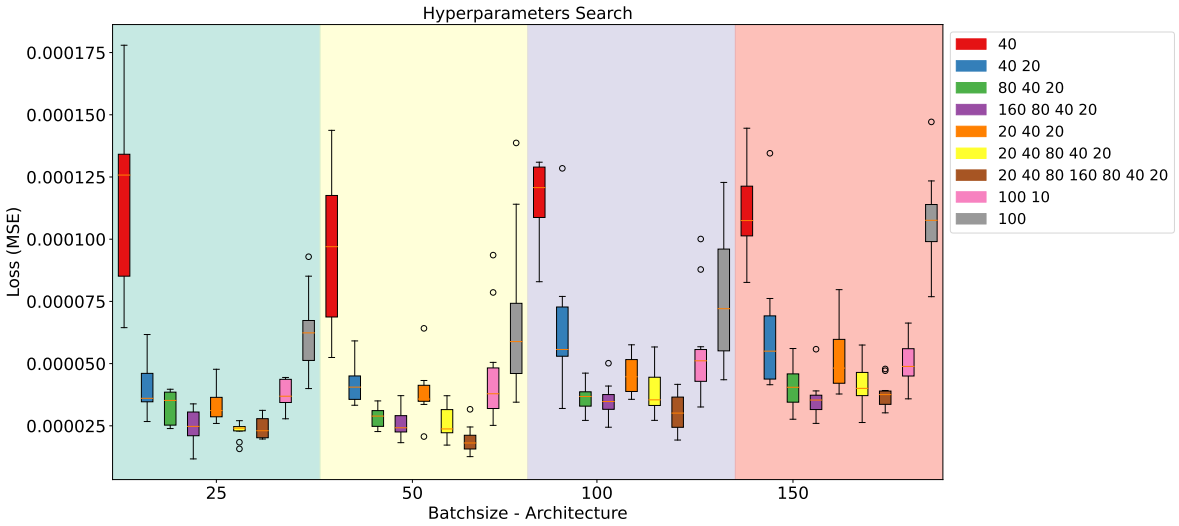

Figure 5.12. Grid Search results - Single coefficient MLP (C_L)

Fig. 5.12 summarizes the results of the grid search performed on the MLP trained on the single aerodynamic coefficient C_L, in this case the preliminary representation in logarithmic scale is not necessary since the aren't results with outliers that have different order of magnitude of the Loss.
We can easily note the different order of magnitude for the loss from the previous results (for this MLP the order of magniture for the los is $10^{-5} \div 10^{-4}$), this is due to the fact that the Mean Squared Error is an absolute difference, so its value depends on the order of magnitude of the aerodynamic coefficient predicted. As already done in Section 5.2, the model with architecture 3, 80, 40, 20, 1 (green boxes in Fig. 5.12) and with a batch size of 100 samples can be chosen as 'best' trade off between computational cost and result accuracy.

The detailed results for this configuration of the single coefficient MLP (C_L) for one of the training sessions performed are represented in Fig. 5.13.

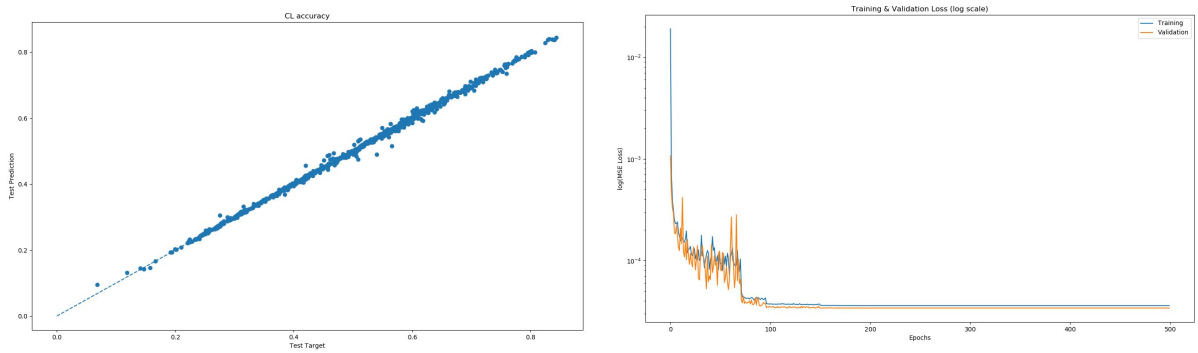

Figure 5.13. Accuracy of the Test prediction (left) and Loss vs Epochs (right)

Double Coefficient MLP - C_L and C_D

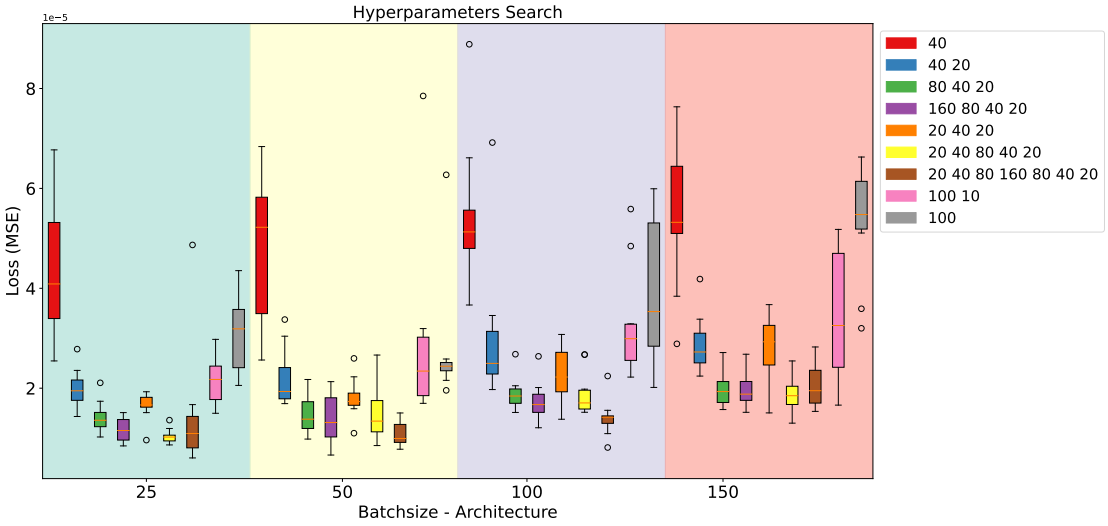

Figure 5.14. Grid Search results - Double coefficient MLP (C_L and C_D)

The first aspect to notice is the order of magnitude of the loss, which is the same of the single coefficient MLP for C_L. This is due to the fact that the overall Loss of the double coefficient MLP is the sum of the Losses of the two output neurons (one for C_L and one for C_D), since the Loss of the C_L is one order of magnitude larger than the C_D one, the contribute of the Loss of the C_D is basically neglected.

For this type of MLP we can choose as best model, in term of performances and computational cost, the architecture with 6 layers of 3, 160, 80, 40, 20 and 2 neurons (purple boxes in Fig. 5.14) with a batch size of 100 samples.
The detailed results for this network configuration are:

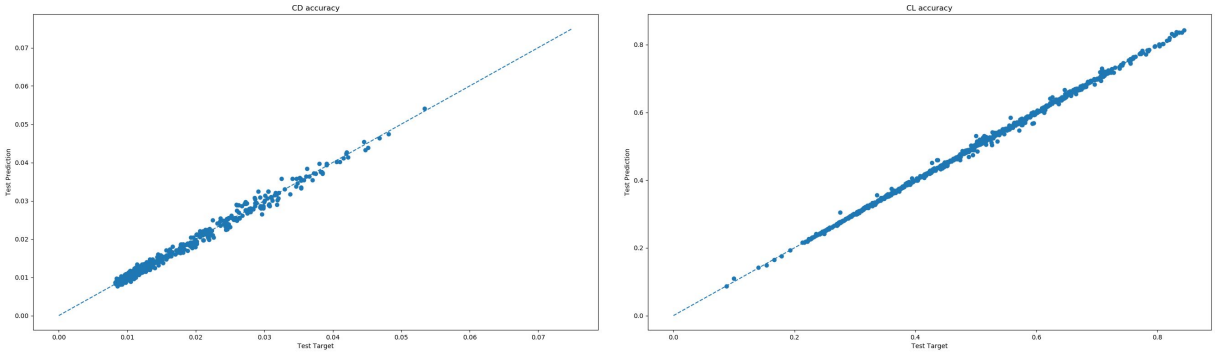

Figure 5.15. Accuracy of the Test prediction for C_D (left) and for C_L (right)

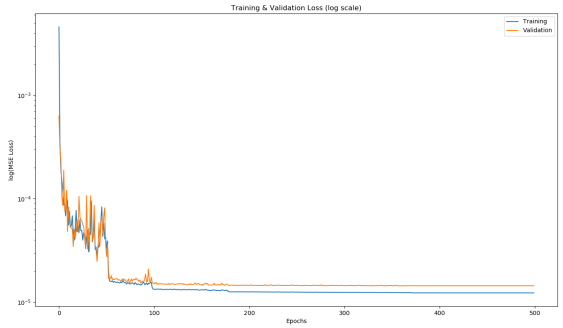

Figure 5.16. Loss vs Epochs, double coefficient MLP

It can be noted that the accuracy of the C_D is lower compared to the C_L one, this is related to the aforementioned difference of order of magnitude for the Losses of the two coefficients. Since the Loss of C_D is much lower compared to the C_L one, the network 'focuses' the training to improve the prediction of the C_L and ignores the C_D.

5.3 Results of the Flow Field Prediction Multi-Layer-Perceptrons

As already mentioned in Section 4.3, the grid searches performed for the flow field prediction MLPs count only the network architecture as variable parameter. The results for this kind of MLP are summarized in box-plots where every box represents a network architecture.

Flow Field Prediction MLP - Grey-Scale

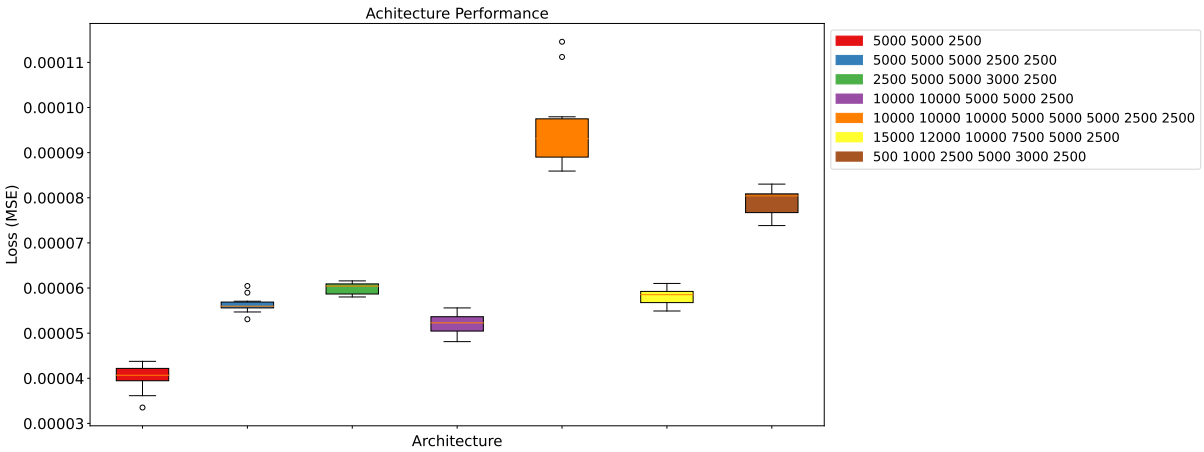

Figure 5.17. Grid Search results - Grey-Scale MLP

The order of magnitude of the loss for this first MLP is $10^{-5} \div 10^{-4}$. The most notable feature of Fig. 5.17 is that, contrary to expectations, the simplest model tested, made of 5 layers with 3, 5000, 5000, 2500 and 2500 neurons (red box in the Fig. 5.17), has achieved the best results in terms of accuracy and, since it's the simplest architecture, it's also the cheapest one in terms of computational cost. This means that an over-complicated architecture, like the one represented by the orange box, can leads to overfitting of the training data, therefore the performance of the network gets worse.
Fig. 5.18 and Fig. 5.19 show the test prediction of the simplest network architecture, computed and saved at different epochs, in order to show the learning process of the network, the last figure shows the target image as reference.
As we can see the model learns very fast, the prediction after only 500 epochs is already very accurate and doesn't change significantly at the end of the training (2000 epochs). This trend is also detectable in Fig. 5.20. This means that, to obtain the very accurate result showed in Fig. 5.19, 2000 epochs of training are

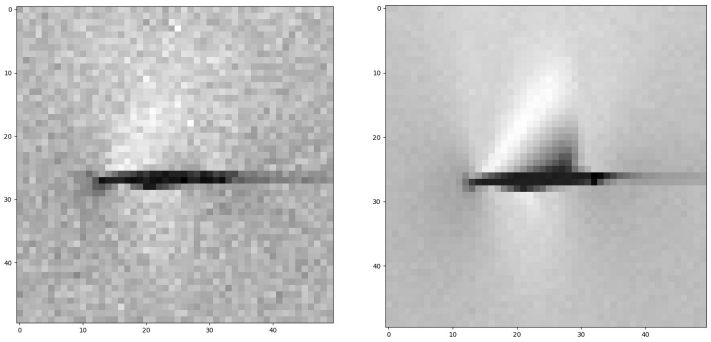

Figure 5.18. Test prediction at the beginning of the training (left) and after 500 epochs (right)

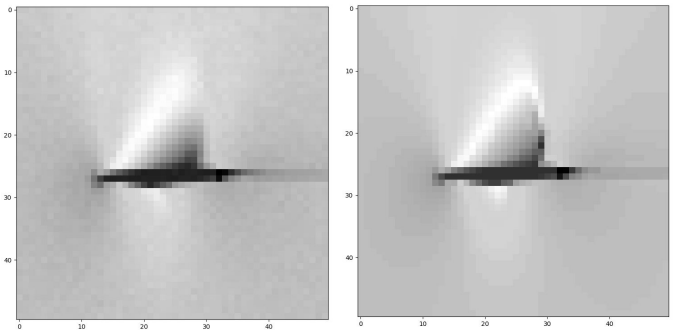

Figure 5.19. Test prediction after the training (left) and target (right)

unnecessary, we can set a number of epochs equal to $500 \div 1000$ to obtain the same result but halving the training time.

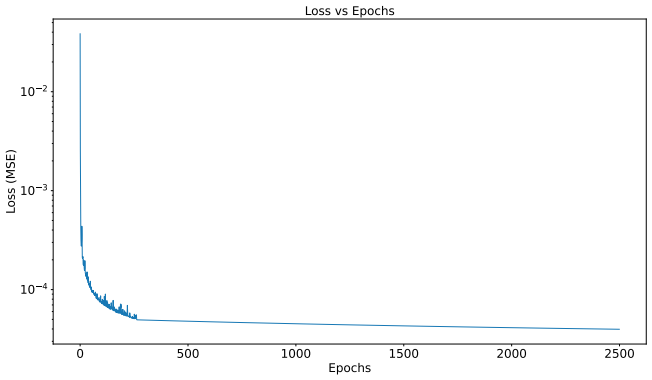

Figure 5.20. Loss vs Epochs - Grey-Scale MLP

Flow Field Prediction MLP - Colored

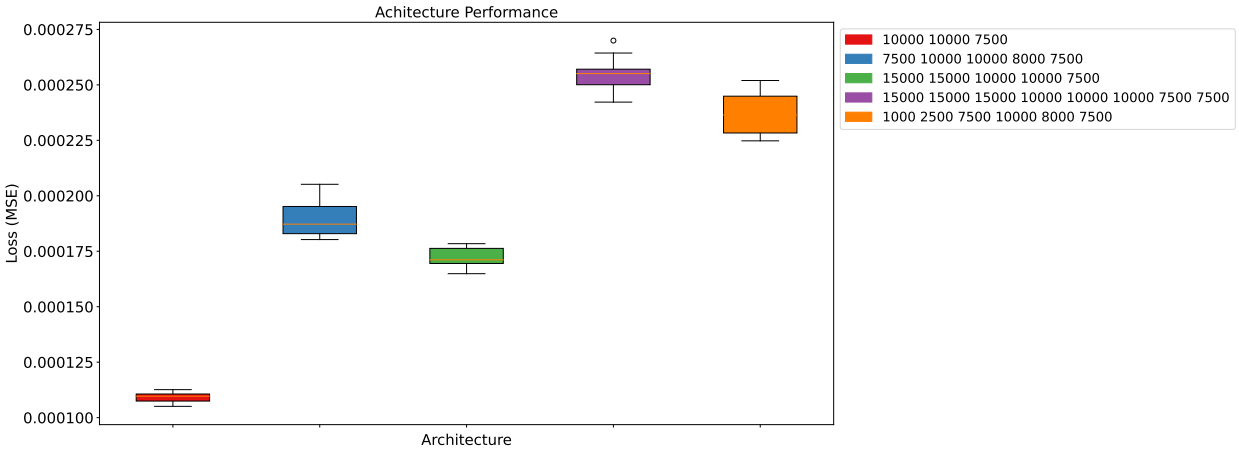

Figure 5.21. Grid Search results - color MLP

Looking at Fig. 5.21 we can see that the order of magnitude of the Loss for this type of MLP is 10^{-4}, the higher order of magnitude is due to the increased complexity of the output that has to be learned. We can see that, as in Section 5.3, the best result is obtained with the simplest architecture (red box in fig 5.21), which consists of 5 layers with 3, 10000, 10000, 7500 and 7500 neurons. The more complex architectures lead to overfitting, therefore their performance gets worse.
The learning history, at the beginning of the training, after 500 epochs and at the end of the training, for the previously described architecture is described in Fig. 5.22 and 5.23.

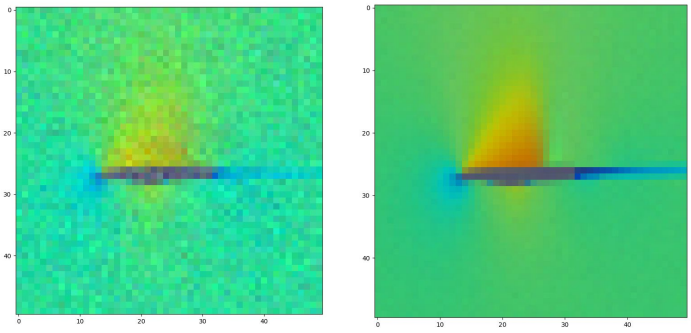

Figure 5.22. Test prediction at the beginning of the training (left) and after 500 epochs (right)

Fig. 5.22 and Fig. 5.23 show the same behaviour of the previous grey-scale MLP, after 500 epochs the model is almost fully trained and the difference between the

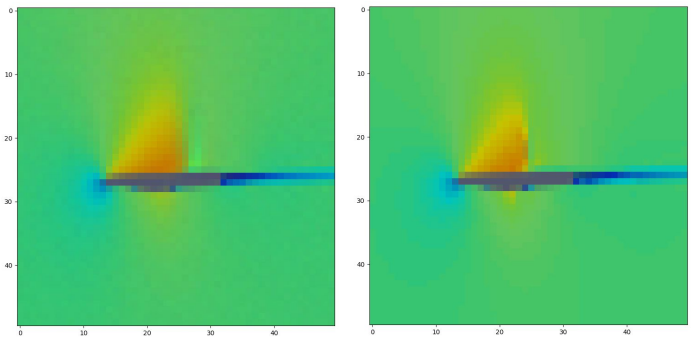

Figure 5.23. Test prediction after the training (left) and target (right)

test prediction at 500 epochs and the test prediction at the end of the training (after 2000 epochs) is negligible. Looking at Fig. 5.24, it shows that the Loss steeply decreases up to 500 epochs and, after $\sim$ 1000 epochs, the slope of the Loss becomes suddenly almost flat.
We can now compare the actual training time necessary for this model with the estimated one in Section 4.3. Previously it was said that a single training of this kind of network could take from one to two hours of computational time; in reality, considering the results obtained, a single epoch of the best model for the colored images MLP takes 2 seconds to be performed (on HPC using one GPU as computing device), if the training lasts 1000 epochs the actual computational time needed to train this network is around 33 minutes.

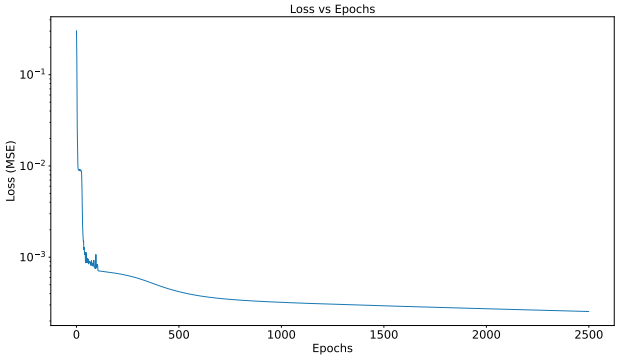

Figure 5.24. Loss vs Epochs - color MLP

Finally, comparing the results of the grey-scale and color MLPs (Fig. 5.19 and Fig. 5.23), we can notice that for the grey-scale model the prediction, outside the supersonic pocket on the upper part of the airfoil, is almost 'flat' and difficult to recognize. On the other hand, the higher contrast and saturation of the colored predictions led to a clearer representation of the flow field.

5.4 Conclusions & Open tasks

All the results presented since now show that the pattern behind the relation between the free stream conditions and the solutions of the CFD simulations, in the form of aerodynamic coefficients or even in the form of the flow field around the airfoil, can be easily learned from a Multi-Layer Perceptron. Moreover we found that the network architectures needed to learn these patterns are quite simple and don't involve overwhelming computational costs. Once the influence of the hyperparameters has been explored and the optimal configuration for the Neural Network has been found, we can use the trained model to compute the inference for the desired set of input features and obtain the prediction of a CFD solution in a few seconds. This leaves a lot of room for improvement for the model, which can be extended to more generalized applications, such as using different airfoils for the simulation and giving as input parameter the airfoil used in order to train the model to recognize the type of airfoil and then to predict the solution of CFD simulation around it. The fluid dynamic problem can also be extended to the 3D domain, for example to predict the pressure distribution over a wing or a fuselage in different flow conditions.
Considering the results obtained for the prediction of the flow field near the RAE2822 airfoil, if we train the model on higher resolution images, or directly on the actual solution in terms of cell-center values, we can obtain a detailed prediction of it. Using this prediction as preliminary result we can compute the gradient of the solution and define a mesh refinement function which can produce a detailed adaptive mesh for the fluid dynamic problem.
Going much deeper, we can try to directly predict the mesh refinement function, using the solution of the CFD simulations as input parameter, and use the 'tailored' refined mesh to extrapolate a more accurate solution by interpolation of the solution on the coarse grid on the refined one.

In conclusion, Machine Learning is an incredible source of applications and as the technology advances and the computation power available increases, the capabilities and possibilities to implement the Neural Networks on large scale problems grows significantly and becomes more and more accessible.

Bibliography

[1] Gmsh website main page. https://https://gmsh.info/.

[2] November 2020 — top500. https://www.top500.org/lists/top500/2020/11/. [Online; accessed 11-May-2021].

[3] Perceptron. https://deepai.org/machine-learning-glossary-and-terms/perceptron. [Online; accessed 13-May-2021].

[4] Siemens simcenter star-ccm+ product main webpage. https://www.plm.automation.siemens.com/global/it/products/simcenter/STAR-CCM.html.

[5] The spalart-allmaras turbulence model — comsol documentation. https://doc.comsol.com/5.6/docserver/\#!/com.comsol.help.cfd/cfd_ug_fluidflow_single.06.093.html?highlight\=spalart\%25E2\%2590\%25A4allmaras. [Online; accessed 27-May-2021].

[6] Su2 website main page. https://su2code.github.io.

[7] S. R. Allmaras, F. T. Johnson, and Spalart P. R. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. In *7th International Conference on Computational Fluid Dynamics*. ICCFD, 2012.

[8] Rami A. Alzahrani and A. Parker. Neuromorphic circuits with neural modulation enhancing the information content of neural signaling. *International Conference on Neuromorphic Systems 2020*, 2020.

[9] B. Aupoix and P. R. Spalart. Extensions of the spalart-allmaras turbulence model to account for wall roughness. *International Journal of Heat and Fluid Flow*, pages 454–462, Vol. 24, 2003.

[10] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[11] Joseph Boussinesq. *Théorie analytique de la chaleur mise en harmonic avec la thermodynamique et avec la théorie mécanique de la lumière*. Gauthier-Villars, 1909.

[12] Jason Brownlee. *Master Machine Learning Algorithms – Discover how they work and implement them from scratch*. Machine Learning Mastery, 2016.

[13] Frederik Bussler. Spiking neural networks — a more brain-like ai. *Bitgrit - AI for All*, 2019.

[14] Ward Cheney. *Analysis for Applied Mathematics*. New York: Springer, 2001.

[15] Langley Research Center Christopher Rumsey. The spalart-allmaras turbulence model. `https://turbmodels.larc.nasa.gov/spalart.html`. [Online; accessed 28-May-2021].

[16] CFD Online Wiki Contributors. Approximation schemes for convective term — cfd online. `https://www.cfd-online.com/Wiki/Approximation_Schemes_for_convective_term_-_structured_grids_-_Common`. [Online; accessed 2-June-2021].

[17] CFD Online Wiki Contributors. Linear eddy viscosity models — cfd online. `https://www.cfd-online.com/Wiki/Linear_eddy_viscosity_models`. [Online; accessed 23-May-2021].

[18] Torch contributors. Cross entropy loss. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`. [Online; accessed 6-June-2021].

[19] Torch contributors. Mse loss. `https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html`. [Online; accessed 6-June-2021].

[20] Torch contributors. Reduce lr on plateau. `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html#torch.optim.lr_scheduler.ReduceLROnPlateau`. [Online; accessed 16-June-2021].

[21] Wikipedia contributors. Activation function — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Activation_function`. [Online; accessed 6-June-2021].

[22] Wikipedia contributors. Spalart–allmaras turbulence model — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Spalart%E2%80%93Allmaras_turbulence_model`. [Online; accessed 06-June-2021].

[23] Wikipedia contributors. Stochastic gradient descent — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Stochastic_gradient_descent`. [Online; accessed 8-June-2021].

[24] Wikipedia contributors. Types of mesh — wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Types_of_mesh`. [Online; accessed 29-May-2021].

[25] P.H. Cook, M. A. McDonald, and M. C. P. Firmin. Aerofoil rae 2822 - pressure distributions, and boundary layer and wake measurements. In *AGARD Advisory Report 138, Experimental Data Base For Computer Program Assessment - Report of the Fluid Dynamics Panel Working Group 04*, chapter 6. NATO - Advisory Group for Aerospace Research and Development, 1979.

[26] Thomas Corke and Hasan Najib. Illinois institute of technology. `https://www.iit.edu/`. [Online; accessed 02-June-2021].

[27] B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, No. 6:793–800, 1934.

[28] Niklas Donges. Gradient descent: An introduction to one of machine learning's most popular algorithms. *Built in*, 2019.

[29] Nelson Dunford and Jacob Schwartz. *Linear operators II. Spectral Theory: Self Adjoint Operators in Hilbert Space*. Interscience Publishers, 1963.

[30] Rohith Gandhi. Introduction to machine learning algorithms: Linear regression. *Towards data science*, 2018.

[31] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

[32] Donald O. Hebb. *The Organization Of Behavior: A Neuropsychological Theory*. New York: John Wiley & Sons, 1949.

[33] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, 2012.

[34] Anthony Jameson. *The Origins and Further Development of the Jameson-Schmidt-Turkel (JST) Scheme*. Department of Aeronautics and Astronautics, Stanford University, CA, 2015.

[35] Migel Jonal. Reynolds averaged navier-stokes (rans) equations derivation and analysis - literature review. `https://skill-lync.com/student-projects/test-15`. [Online; accessed 21-May-2021].

[36] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*. ICLR, 2015.

[37] Daniel Kobran and David Banys. Gradient descent — ai wiki. `https://docs.paperspace.com/machine-learning/wiki/gradient-descent`. [Online; accessed 8-June-2021].

[38] Simeon Kostadinov. Understanding backpropagation algorithm. *Towards data science*, 2019.

[39] Randall J. Leveque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002.

[40] J. M. McDonough. Reynolds decomposition. In *Introductory Lectures on Turbulence*, chapter 1, Fundamental considerations. Departments of Mechanical Engineering and Mathematics, University of Kentucky, 2004.

[41] Boudjelal Meftah, Olivier Lézoray, Soni Chaturvedi, Aleefia A Khurshid, and Abdelkader Benyettou. Image processing with spiking neuron networks. In *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, pages 525–544. Springer, 2013.

[42] Glenn Research Center Nancy Hall. Navier-stokes equations — 3 dimensional

- unsteady. https://www.grc.nasa.gov/www/k-12/airplane/nseqs.html. [Online; accessed 16-May-2021].

[43] S. R. Nandakumar, R. Kulkarni Shruti, V. Babu Anakha, and Rajendran Bipin. Building brain inspired computing systems. *IEEE Nanotechnology Megazine*, page 20, September, 2018.

[44] Micheal A. Nielsen. How the backpropagation algorithm works. In *Neural Networks and Deep Learning*, chapter 2. Determination Press, 2015.

[45] Allard Overmeen. https://www.bronkhorst.com/int/blog-1/what-is-the-difference-between-laminar-flow-and-turbulent-flow/. https://www.bronkhorst.com/int/blog-1/what-is-the-difference-between-laminar-flow-and-turbulent-flow/. [Online; accessed 02-June-2021].

[46] Sharad N. Pachpute. Basics of cfd modeling for beginners. https://cfdflowengineering.com/basics-of-cfd-modeling-for-beginners/#:~:text=Computational\%20fluid\%20dynamics\%20(CFD)\%20is\%20the\%20science\%20of\%20predicting\%20fluid,mass\%20transfer\%20and\%20species\%2C\%20etc. [Online; accessed 14-May-2021].

[47] C. C. Paige and M. A. Sounders. Solution of sparse indefinite systems of linear equations, 1975.

[48] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[49] Siva prasad. Deriving the reynolds averaged navier-stokes equations - literature review. https://skill-lync.com/projects/Deriving\-the\-Reynold\-s\-Averaged-Navier\-Stokes-RANS-Equations\-Literature-Review\-99705. [Online; accessed 20-May-2021].

[50] Baidurja Ray, Rajesh Bhaskaran, and Lance R Collins. Introduction to cfd basics, 2012.

[51] Osborne Reynolds. On the dynamical theory of incompressible viscous fluids and the determination of the criterion. *Philosophical Transactions of the Royal Society of London A.*, page 123–164, 1895.

[52] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, pages 533–536, No. 323, 1986.

[53] scikit-learn developers. Neural network models (supervised). https://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Online; accessed 4-June-2021].

[54] Avinash V. Sharma. Understanding activation functions in neural networks. *The Theory of Everything*, 2017.

[55] Sagar Sharma. Activation functions in neural networks. *Towards data science*, 2017.

[56] M. L. Shur, M. K. Strelets, A. K. Travin, and Spalart P. R. Turbulence modeling in rotating and curved channels: Assessing the spalart-shur correction. *AIAA Journal*, pages 454–462, Vol. 34, No. 5, 2000.

[57] John W. Slater. Rae2822 transonic airfoil. `https://www.grc.nasa.gov/WWW/wind/valid/raetaf/raetaf.html`, 30 September 1998 — Updated on 17 December 2002. [Online; accessed 23-April-2021].

[58] John W. Slater. Rae2822 transonic airfoil:study #1. `https://www.grc.nasa.gov/WWW/wind/valid/raetaf/raetaf.html`, 30 September 1998 — Updated on 17 December 2002. [Online; accessed 20-June-2021].

[59] Joseph Smagorinsky. General circulation experiments with the primitive equations. *Monthly Weather Review*, pages 99–164, 1963.

[60] P. R. Spalart and S. R. Allmaras. *A One-Equation Turbulence Model for Aerodynamic Flows*. Recherche Aerospatiale, 1994.

[61] D. B. Spalding. A novel finite-difference formulation for differential expression involving both first and second derivatives. *International Journal for Numerical Methods in Engineering*, 4:551–559, 1972.

[62] Sergios Theodoridis. Online learning: the stochastic gradient descent family of algorithms. In *Machine Learning (2nd edition)*, chapter 5, pages 179–251. Elsevier, 2020.

[63] Sakshy Tiwar. Activation functions in neural networks. `https://www.geeksforgeeks.org/activation-functions-neural-networks/`. [Online; accessed 8-June-2021].

[64] Eleuterio F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg, 1990.

[65] Airfoil Tools users. Rae 2822 airfoil (rae2822-il) — airfoil tools. `http://airfoiltools.com/airfoil/details?airfoil=rae2822-il`. [Online; accessed 21-June-2021].

[66] Airfoil Tools users. Rae 2822 airfoil (rae2822-il) xfoil prediction polar at re=100,000 ncrit=9 — airfoil tools. `http://airfoiltools.com/polar/details?polar=xf-rae2822-il-100000`. [Online; accessed 19-June-2021].

[67] Various. *AGARD Advisory Report 138, Experimental Data Base For Computer Program Assessment - Report of the Fluid Dynamics Panel Working Group 04*. NATO - Advisory Group for Aerospace Research and Development, 1979.

[68] H Versteeg and W Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method (2nd Edition)*. Pearson, 2007.

[69] Julius von Kügelgen. On artificial spiking neural networks: Principles, limitations and potential.

[70] Sutherland William. The viscosity of gases and molecular force. *Philosophical Magazine Series 5*, pages 507–531, 1893.

[71] Christian Wollblad. Your guide to meshing techniques for efficient cfd modeling. `https://www.comsol.com/blogs/`

your-guide-to-meshing-techniques-for-efficient-cfd-modeling/. [Online; accessed 30-May-2021].

[72] Dannis Yoder. Wind-us — version 4.0. https://www.grc.nasa.gov/www/winddocs/, Last Update 30 September 2016. [Online; accessed 20-June-2021].